420-TP-007-001

Planning Workbench Detailed Design for the ECS Project

Technical Paper

September 1995

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Jolyon Martin, Staff Engineer 9/25/95
EOSDIS Core System Project

SUBMITTED BY

Karin Loya, PDPS Release A Manager 9/25/95
EOSDIS Core System Project

Hughes Information Technology Corporation
Landover, Maryland

This page intentionally left blank.

Abstract

This document describes the Release A Delta Design Review (DDR) Design for the SDPS Planning Workbench Computer Software Component (CSC) of the Planning subsystem.

Keywords: Planning, SDPS, Release A, OMT, PDPS

This page intentionally left blank.

Contents

1. Introduction

Tables

1. Introduction

1.1 Purpose

This technical note describes the detailed design of the Planning Workbench component of the Planning Subsystem. The basis for the material in this document is the Release A SDPS Planning Subsystem Design Specification for the ECS Project [305-CD-010-001]. This note presents the design at a more detailed level than the previous document, especially with respect to how the Planning Workbench is implemented within the framework of the Hughes Planning and Scheduling class libraries (Delphi).

The numbering system used within the document is the same as the Planning Design Specification Document. Some introductory and overview material has been included from that document in order to provide some more context to the design. For a complete picture of the Planning Subsystem design, the reader is encouraged to refer back to Planning Design Specification Document.

The material presented here will be incorporated into the next release of the entire Planning Design Specification Document, i.e., that for Release B IDR.

The text that is new to this document has been distinguished from that from the Planning Design Specification Document by side-bars.

The material presented in this document is incomplete in two regards. The OMT consistency checking scripts have not been verified to confirm consistency between the diagrams and the event traces. In addition, the data dictionary has not been updated to reflect the most recent changes.

1.2 Organization

This paper is organized as follows:

Section 1 provides information regarding the purpose and organization of this document.

Sections 4 contain the structure of the Planning Workbench component of the Planning subsystem.

Questions regarding technical information contained within this Paper should be addressed to the following ECS and/or GSFC contacts:

- ECS Contacts
  - Jolyon Martin, Staff Engineer, (301)925-0436, jolyonm@eos.hitc.com
- GSFC Contacts
  - Steve Kempler, PDPS Manager, (301)286-7766, steven.j.kempler@gsfc.nasa.gov

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Landover, MD 20785

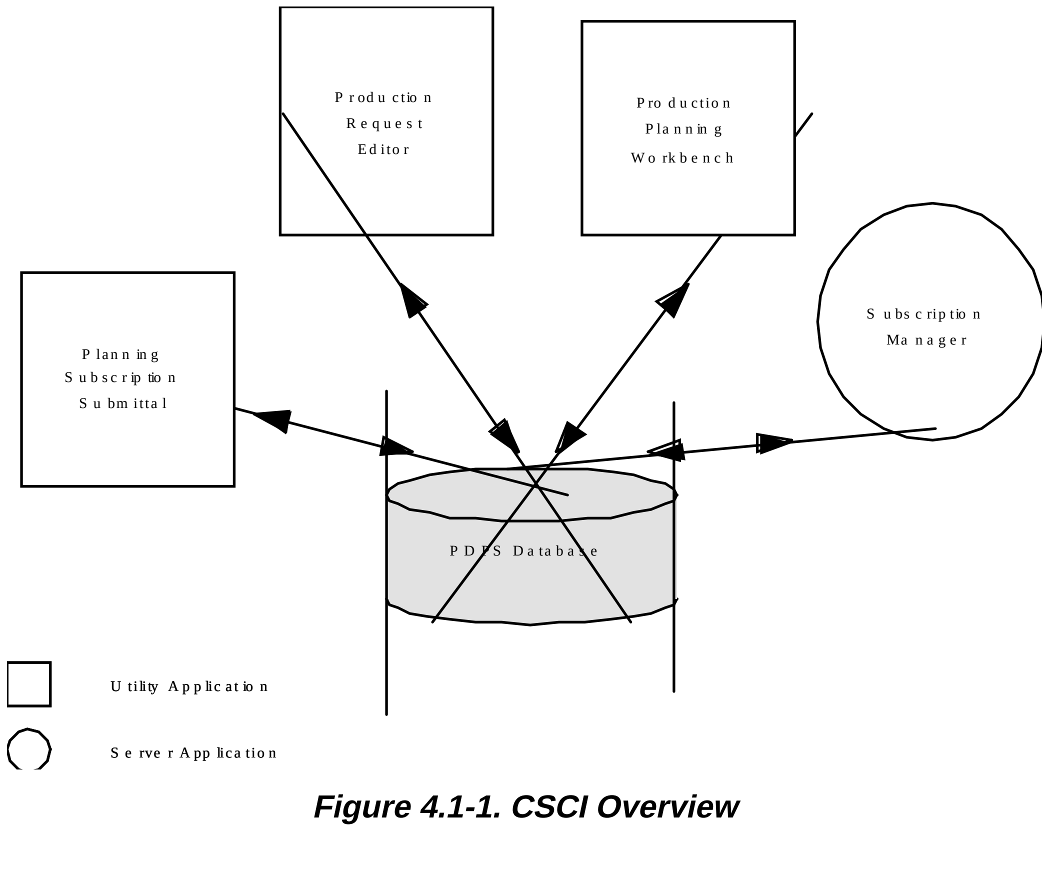
This page intentionally left blank.

4. PLANG - Production Planning CSCI

4.1 CSCI Overview

The Production Planning CSCI consists of a number of utilities and server applications as shown in Figure 4.1-1. The distinction between these two classes of programs is useful when describing the software design:

- Utilities are programs that perform well defined tasks and are invoked at the operator's requests. A utility is usually exited when the task is complete; the system does not depend on the utility being active at any given time. As an example, a mail editor is a common utility program, invoked whenever a user wishes to generate a mail message.
- Servers are programs that perform ongoing tasks and have to be active at all times for the subsystem to carry out its allocated functions. An example here is the mail daemon; the mail won't be delivered unless the daemon is running.



Production Request Editor
Production Planning Workbench
Subscription Manager
Planning Subscription Submittal
PDPS Database
Utility Application
Server Application

Figure 4.1-1. CSCI Overview

The allocation of the Planning subsystem capabilities to distinct applications accounts for the need to provide distinct interfaces for distinct activities, and to restrict access to planning functions to classes of users. The applications define an integrated set of tools with well-defined functions to supplement the COTS components.   The allocation of capabilities to applications within the Planning CSCI also accounts for the need to permit independent development of the components as the capabilities of the ECS evolve. The purpose of each application is briefly outlined below.

4.1.1 PDPS Database

At the heart of the Planning CSCI is the Planning and Data Processing System (PDPS) Database (and the implied database server). This provides the persistent storage for data and facilitates the sharing of this data between the applications. The database also provides security, fault tolerance, and marshals request for concurrent access to data. The objects which are built from the persistent data within the PDPS database are indicated within the object design. The schema for the database has been driven from the object model and is described in the SDPS Database Design and Schema Specification CDRL.

4.1.2 Production Request Editor

This application allows the user to submit production requests that describe the data products to be produced. The application uses the PGE descriptions (profiles) entered during AI&T in order to work out the tasks - Data Processing Requests - that in sum meet the request. The application provides the capabilities to add, modify, and delete Production Requests, as well as review and modify the resulting Data Processing Requests. The production request editor is identified as a distinct application and separate from the workbench in anticipation that defining production requests will be a discrete activity, unrelated to the “planning” of these events.

The Production request editor may also be used by authorized operations staff to schedule Data Processing Requests derived from a production request directly to the Data Processing subsystem.

4.1.3 Production Planning Workbench

The application is used to prepare a schedule for the production at a site, and forecast the start and completion times of the activities within the schedule. These functions provided by the workbench include the following high-level activities:

- Candidate Plan Creation -- from the production requests prepared by the Production request Editor;
- Plan Activation -- activating a candidate plan;
- Updating the Active Plan -- feedback from the processing into the active plan; and
- Canceling/Modifying the Active Plan.

As described previously, activating a plan entails rolling a portion of a selected plan into the AutoSys COTS. This “schedule” is then managed within the Data Processing subsystem. The forecast times generated within the planner are used to set up operator alerts that would make the operator aware of gross departures from the predicted schedule. The production planning workbench can periodically update it's predictions using feedback from the AutoSys.

4.1.4 Planning Subscription Editor

This application provides the capabilities required to submit subscriptions to the Data Servers responsible for the storage of ingested data. Registration of a subscription at a Data Server is required for the Planning CSCI to receive notification when data arrive within the ECS. At this Release, the submission of subscriptions will be managed as an operator initiated activity, although this may be automated at a later date.

4.1.5 Subscription Manager

The Subscription Manager is used to manage the receipt of a subscription notification from the Data Server. Subscription notification is used to notify Planning on the arrival of input data required by a given PGE. The Subscription Notification contains Universal References (URs) which are pointers to the data objects stored in the Data Server. The Subscription Manager updates the PDPS database to indicate when data become available. When all input data for a Data Processing Request is available, the job defined for that Data Processing Request is released within the Data Processing subsystem.

4.3 CSCI Object Model

The object model from the Planning Design Specification Document has been updated to better reflect the full Delphi implementation of the Planning Workbench. The complete object model for the Planning Subsystem was decomposed into a number of views. Each view has a close mapping to one of the Planning CSCs, or illustrates an important aspect of a CSC. The object models pertinent to the Planning Workbench are reproduced here some additions and modifications have been made in order to give a better high level picture of the plan generation aspects.

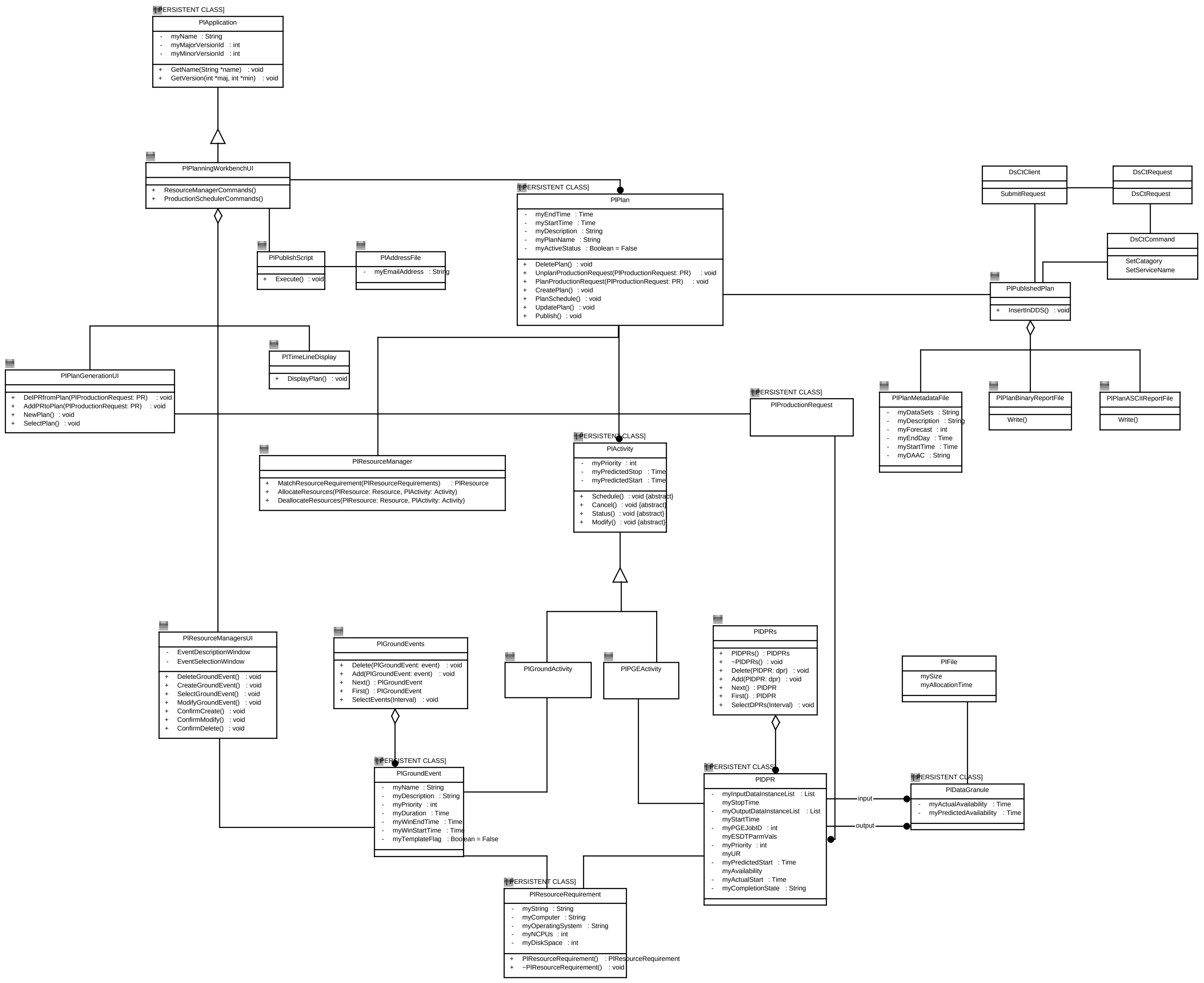
4.3.4 Production Planning View

This view (Figure 4.3-4) describes the classes used in the creation and management of plans. The design presented here for a number of key classes is at a high level of abstraction. This level of abstraction is offered to explain the design without going into detail about the planning framework (Delphi) which supports the plan generation. The planning framework within the Production Planning CSCI is documented in detail within the Planning Object Library CSC, section 4.6.5. That section contains object models, class descriptions for the framework and maps the objects used in framework to the abstractions presented here.

The diagram introduces the following key classes:

- PlPlanningWorkbenchUI: This class is an abstraction for the non-graphical aspects of the user interface to the planning workbench application. The interface will be developed with a suitable GUI builder tool. The interface is shown to be
- PlPlan: This class represents an abstraction for a production plan. The class describes the metadata that will be stored for a plan within the PDPS database.
- PlActivity: This class describes an item within a plan. The activity class is a base class within a specialization hierarchy describing the different activities which occur in the production plan.

Figure 4.3-4 Production Planning Object Model



- PlPGEActivity: This class is a specialization of the PlActivity class. The class describes a Data Processing Request - a run of a PGE - within the plan.
- PlGroundActivity: This class is a generalization of the PlActivity class. The class describes a Ground Event within the plan.
- PlGroundEvent: This class describes a Ground Event which is recorded in the PDPS database. A Ground Event marks the allocation of resources to some non-production task such as maintenance.
- PlResourceManager: This class represents an abstraction for the resource management capabilities used when generating a plan, describing the operations required to match resource requirements of an activity to the available resources, and to allocate the resource for the activity. This class has a close mapping to the 'scheduler classes' within Delphi, which are the components that implements the 'planning algorithm' responsible for the plan generation. To avoid terminology confusion with the Data Processing Subsystem, and the DpPrScheduler which is the interface class used by Planning to seed information into the AutoSys Job Scheduling Engine the class name of PlResourceManager is retained here.
- PlFile: This class represents an input or output granule which is consumed or produced by a DPR. The file is allocated to a Disk resource when the Activity is allocated to a DPR.
- PlResourceRequirement: This class describes the resource requirements for a task. These requirements may then be matched against the actual resources available.
- PlPublishedPlan: This object encapsulates the methods required to insert externalized formats of the plan into the document data server.

The interactions of these classes are described within the following scenarios:

- 4.5.6 Adding/Modify/Deleting a Ground Event
- 4.5.7 Creating a Plan
- 4.5.8 Deleting a Plan
- 4.5.9 Publishing a Plan

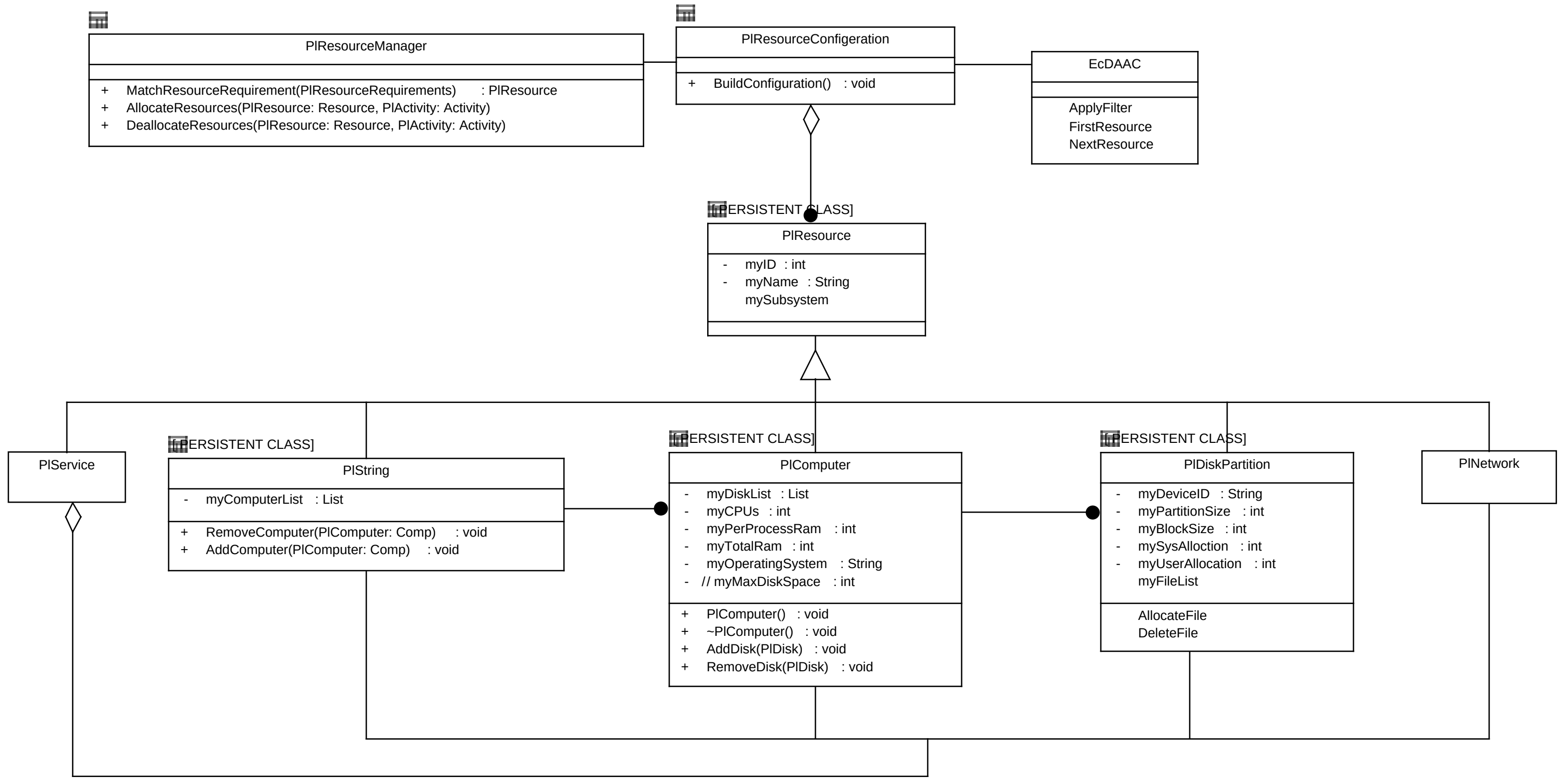
4.3.5 Resource Management View

This view (Figure 4.3-5) describes the classes used in the description of the production resource configuration. The MSS provides the planning subsystem (and all subsystems) with resource configuration information. The view is a continuation from the Production Planning Workbench view, presented separately to reduce the complexity of the previous diagram.

The diagram introduces the following key classes:

- PlResource: This class is the base class in a generalization heirachy describing the production resources.
- PlString: A string describes the logical collection of a number of resources which may be allocated for an instruments processing needs.
- PlComputer: This class describes the production computers

Figure 4.3-5. resource Management Object Model



- PlDiskPartition: This class describes the disk resources for data production. The DiskPartition resource tracks the PlFiles allocated to it. When allocated to a DiskPartition the class first looks to see if it has enough space to allocate the File without deleting any other files on the same partition. If there is then the file is allocated and the total disk space is reduced by the appropriate amount. If there is not enough space then the DiskPartition looks to see if it has Files that are not in use at the time of allocation, and deletes the oldest of these in order to free space. Note that this is just a model of the resources and the file management scheme used within the data processing system. There are no physical files being created or deleted at this time, just object representations of them.
- PlNetwork: This class describes the network resources.
- PlService: This class describe a high level aggregation of resources as allocated to a system service, such as L0 acquire, or insert (archive) of data.
- MsDAAC: This class provides MSS configuration information to the planning subsystem.

The resources classes managed within the Planning Workbench have been supplemented since the CDR version of the Design Specification Document in order that ground events may be recorded against all ECS components. The Planning Subsystem may thus account for the dependencies on non-production resources such as the Data Server availability etc. The following simplification ap plies. The aggregation shown between a service and it’s component resources will not be imple mented at Release A. In other words, the Planning Subsystem won’t be able to associate the fact that a ground event on a Ingest disk, or CPU will result in a higher level service, such as the CERES L0 acquire service, to be unavailable. In this case the Resource Planner (operations role) will have to make the association and enter an additional ground event against the appropriate service to record it’s non-availability to data production.

The interactions of these classes are described in the following scenario

- 4.5.10 Building the Resource Configuration

4.3.6 Plan Activation View

This view (4.3-6) again shows more detail for the Planning Workbench. It explicitly shows the Data Processing subsystem interface.

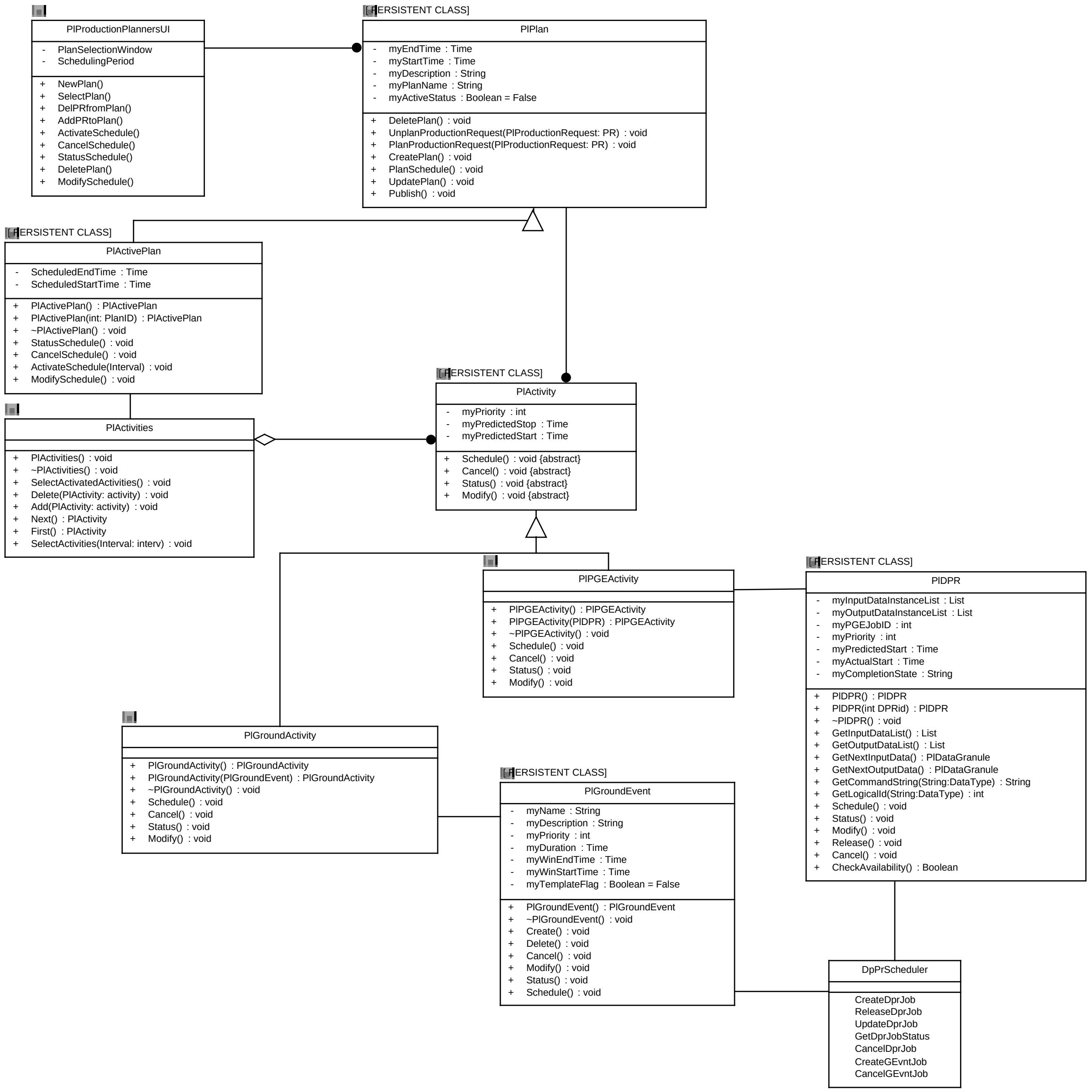
The diagram introduces the following key classes:

- PlProductionPlannersUI: This class is an abstraction for the user interface to the planning workbench application. The interface will be developed with a suitable GUI builder tool. The class does describe the basic operations that are provided from the interface.
- PlActivePlan: This class is the specialization of the PlPlan class and contains the methods to manage the activation, canceling, and statesman of a plan.
- DpPrScheduler: This class is the interface to the Data Processing subsystem for entering jobs into the job scheduler.

The interactions of these classes are described in the following scenario:

- 4.5.11 Activating a Plan,
- 4.5.12 Canceling a Plan, and
- 4.5.13 Statusing/Updating a Plan.

Figure 4.3-6. Plan Activation Object Model



4.5 PLANG Dynamic Model

The PLANG Dynamic model presents a number of scenarios and event traces that describe the key interactions of the classes participating in the various components of the Planning subsystem.

Those scenarios which pertain to aspects of the Planning Workbench are described here.

4.5.6 Ground Event Scenario

4.5.6.1 Abstract

This scenario describes the system response to a resource manager adding / modifying / deleting a ground event within the PDPS database. Ground events describe the allocation of a resource to a non-production task such as maintenance. In the majority of cases, these ground events will be de fined well in advance of the production planning, and so a replanning of the production schedule is not part of this standard scenario. If the ground event is entered within a time period that is part of the activated schedule then, in order to have that event figure within the schedule, a replan will be required the resource manager (operations staff role) is required to communicate the need for a replan with the production scheduler (operations staff role) in order for that event to be scheduled.

Note: The description of a ground event allows a window of opportunity to be defined together with a duration. (So that the production can have some flexibility as to when to plan an event to achieve efficient resource usage). The actual time of the event will only be determined when the plan is generated, unless the duration is the same length of time as the window of opportunity.

4.5.6.2 Interfaces With Other Subsystems and Segments

None.

4.5.6.3 Stimulus

The resource manager initiates the Production Planning Workbench in order to enter a ground event.

4.4.6.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlResourceManagersUI
- PlGroundEvents
- PlGroundEvent

4.5.6.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running.

4.5.6.6 Ending State

No change in PDPS database server state.

4.5.6.7 Scenario Description

1.- The user starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17).

Thread 1: Addition of a ground event

2.- The user is presented with a function to add a new event, this initiates a forms type window (Ground Event Editor) in which that ground event can be described, the user populates the fields within the form, for event description, window of opportunity, duration, resource requirements etc.

3.- The user confirms the description is complete and the Ground Event is created and recorded within the PDPS database. (Simple validation of the Ground Event will be performed to ensure the start date is before the end date etc.).

4. The timeline display of resources will be updated to show the allocation.

Thread 2: Modification of a ground event.

5.- The user is presented with a function to modify an event described within the PDPS database, this initiates a window (Event Selection Window) which provides a list of the current defined events. The facility to reduce the list by limiting the search to a given time range is provided.

6.- The user selects a ground event from the list, which then populates the fields in the Event Description Window.

7. The user modifies the description and confirms the modification of the Ground Event.

8. The timeline display of resources will be updated to show the allocation.

Thread 3: Deletion of a ground event.

9.- The user is presented with a function to delete an event described within the PDPS database, this initiates a window (Event Selection Window) which provides a list of the current defined events. The facility to reduce the list by limiting the search to a given time range is provided.

10. The user selects a ground event from the list, which then populates the fields in the Event Description Window.

11. The user confirms the deletion of the Ground Event.

12. The timeline display of resources will be updated to show the allocation.

4.5.6.8 Event Trace

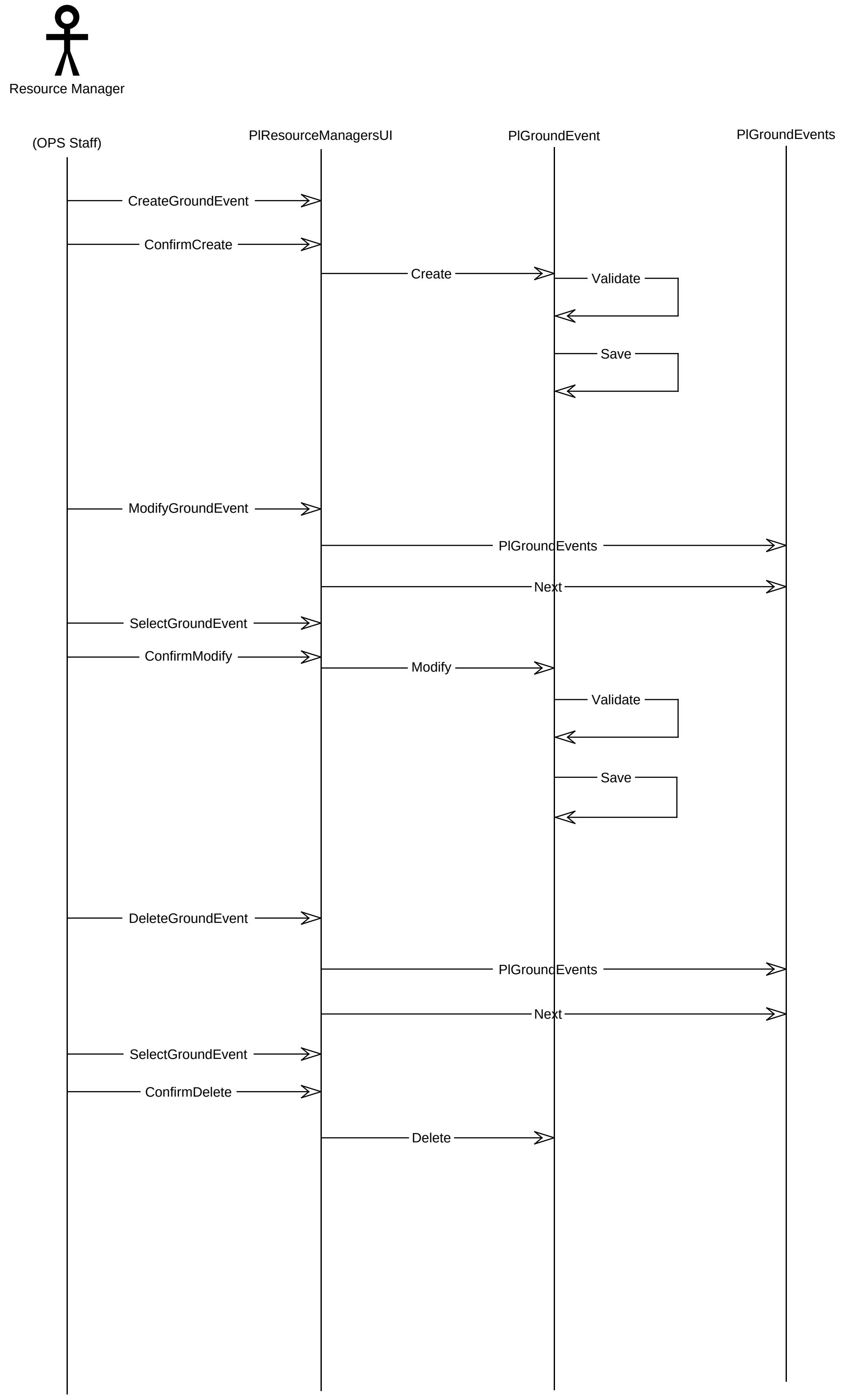
See Figure 4.5-9.

4.5.7 Plan Creation Scenario

4.5.7.1 Abstract

This scenario describes the creation of a plan within the Production Planning Workbench.

Figure 4.5-9. Ground Event Entry Event Trace



This scenario presents an abstract representation of the activities that occur within the Production Planning Object Library. The full detail of generating a plan is fairly complex, and intimately tied to the Production Planning Object Library. This scenario is presented to describe the process at a reasonable level of detail. For fuller description please refer to the Production Planning Object Li brary CSC section (section 4.6.5).

4.5.7.2 Interfaces With Other Subsystems and Segments

None.

4.5.7.3 Stimulus

The production scheduler initiates the Production Planning Workbench in order to generate a new plan.

4.5.7.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlPlan
- PlGroundEvents
- PlGroundEvent
- PlGroundActivity
- PlDPRs
- PlDPR
- PlPGEActivity
- PlResourceManager

4.5.7.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running.

4.5.7.6 Ending State

No change to system state.

4.5.7.7 Scenario Description

1. The user starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17). The user specifies the time period for which to generate a plan within the GUI and initiates the creation of a plan object.
2. The plan object determines which ground events are defined within the duration of the planning time-period, by creating an instance of the PlGroundEvents collection object.
3. The plan object iterates through the ordered list of ground events, creating an activity to fulfill each event
4. The activity is allocated to the appropriate resources by the resource manager, thus setting out a timeline of when the resources are unavailable for production.

5. The user then specifies the Production Requests which are to be included in the plan.

6.≠ The Data Processing Requests which are associated to the Processing Request, and within the time period of the plan are determined from the PDPS database using the PlDPRs collection class.

7.≠ The PlResourceManager object iterates through the list of Data Processing Requests, creating an ordered list of activities which describe the DPRs.

The sort algorithm orders the list of data processing requests based on the data arrival times and DPR dependencies.

7.1 The sort algorithm determines the first DPR item from the list for which all the input products are available. This item, DPR_1 is tagged with the earliest time that it may be scheduled by using the latest time from the predicted availability times of the input data .

7.2 The algorithm iterates over the output products of DPR_1 and (for the purposes of the sort algorithm) declares the predicted time of the output data to be the same as earliest schedule time for DPR_1 (i.e. assumes that the DPR is instantaneous).

7.3 The algorithm determines whether these outputs are the input to any other DPRs (call this DPR_2). For DPR_2 the algorithm determines if all other dependent DPRs scheduled have been scheduled. This is true if all the input granules have a declared predicted availability time. If true then DPR_2 is added to the ordered list after any DPR with the same time. The recursion continues with the output of DPR_2.

7.4 When the end of the chain is reached or a DPR with missing inputs is found then the sort algorithm continues through the list of Data Processing Requests.

The sort algorithm will determine if there are missing dependencies. This could be due to a Production Request for a 'lower level' PGE not being included in the specified list of Production Requests to be scheduled. These errors will be report to the operator.

Note: it is likely that the ordering efficiency may be affected by the way the output data types are listed in the PGE profile. In this case it will be necessary to determine the optimum order for that list in order that an efficient DPR planning will be achieved.

8. The activity is allocated to the appropriate resources by the resource manager.

The allocation scheme for the DPRs has been designed to simulate the sequence in which the Data Processing Subsystem and the Job Scheduler within that Subsystem will perform the activities, as well as to order the activities in an efficient manner.

8.1 The ordered set of activities as prepared in step 7 of this scenario are considered in turn. When there are more than one DPRs that may be scheduled at the same time within the list, the highest priority activity with no outstanding dependencies is chosen.

8.2 The PlResourceManager class determines the set of computers on which this activity may be scheduled (by matching the resource requirements with the resource pool). The allocation is tried against all computers within this set

8.3 The earliest starting time of the DPR (which is determined from the latest availability time for the input data) is used as a starting point in trial allocation of the DPR to the resource. The earliest point at which the computer, and all the other required resources of the DPR at a time after the earliest starting time of the DPR is determined.

8.4 A trial allocation of the DPR is made to the computer, the resource is able to "shift" any ground event within it's window of opportunity in order to schedule the DPR.

8.5 A cost function is calculated in order to generate a preference for one of the resources in the resource set. This is based on the cost of "data transfer" from one computer to another, compared to the "time of data on disk" within the computer. The precise determination of the cost function is TBD, although it is clearly a function of file size. The goals are to allocate chains of DPRs to the same computer to minimize the number of data hops within the processing.

8.6 The DPR is allocated to the "cheapest" resource within the resource set. The DPR is allocated to the CPU and the data files are allocated to the disk. The same file allocation scheme is used as within the Data Processing Subsystem. This is that the data are not deleted from the disk (within the allocation model) until that space is required by some other file, and at that point a first in first out approach is used. This procedure is followed so that data produced as output to some DPR may be used as input to some dependent DPR without going through a staging, destaging cycle.

8.7 The output file prediction times are updated.

9. The schedule for the DPR executions is displayed.

4.5.7.8 Event Trace

See Figure 4.5-10.

4.5.8 Deleting a Plan Scenario

4.5.8.1 Abstract

This scenario describes the deletion plan from the Production Planning Workbench. This function does not delete the associated DPRs or Ground Events from the PDPS database, since they may be associated to other plans, and are also maintained for a set period such that reports may be gener ated against their completion status.

4.5.8.2 Interfaces With Other Subsystems and Segments

None.

4.5.8.3 Stimulus

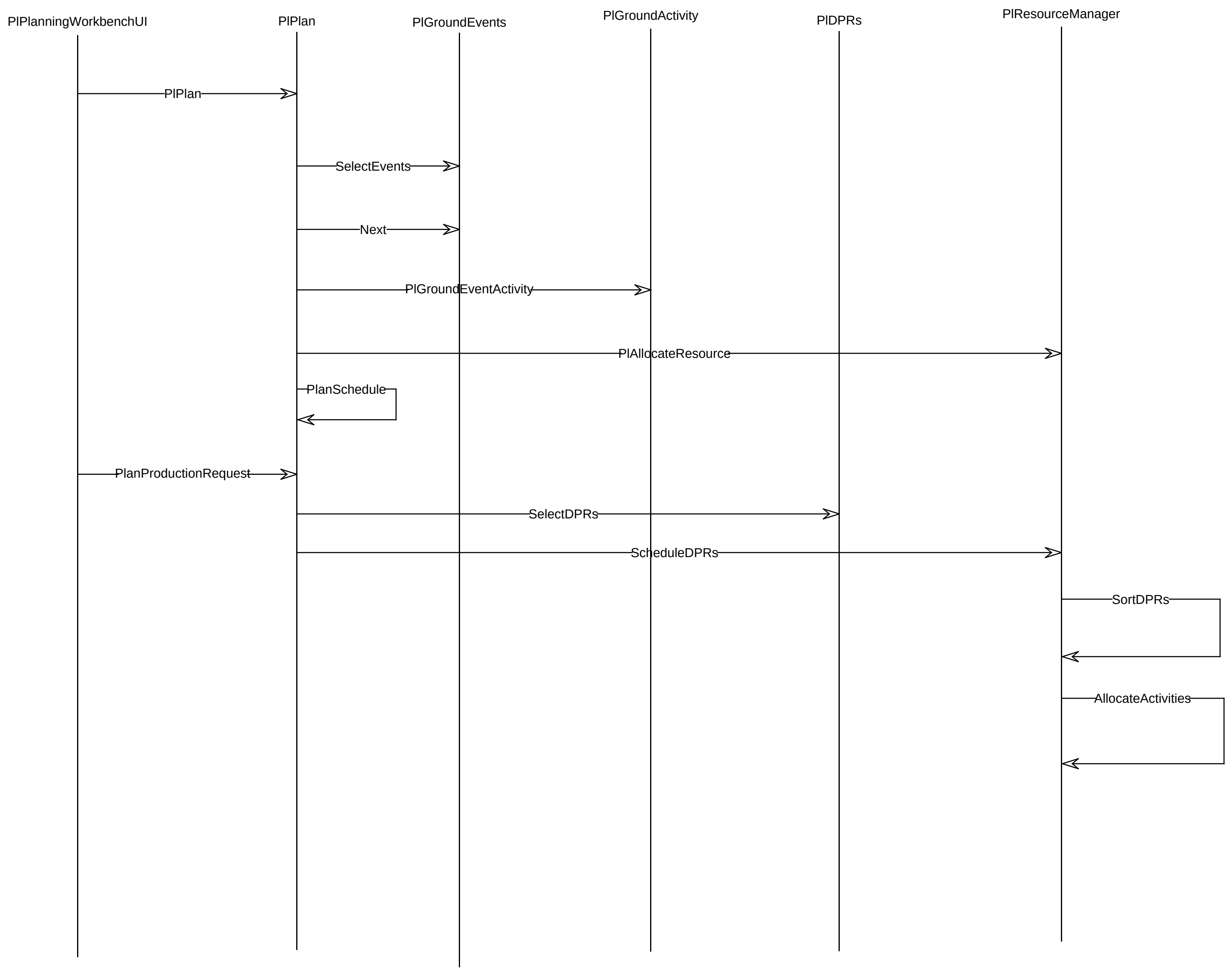
It's assumed that the Production Planning Workbench application is running. The user decides to delete the production schedule from the workbench.

4.5.8.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlPlan
- PlPGEActivities
- PlPGEActivity

Figure 4.5-10. Plan Creation Event Trace



4.5.8.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running. The Production Workbench application is running and that a plan has been selected.

4.5.8.6 Ending State

No change in PDPS database server state. The plan is deleted.

4.5.8.7 Scenario Description

1. The operator initiates the command to delete the displayed plan.
2. The plan object creates an ordered list of the activities within the plan.
3. The plan object iterates through the activities and invokes the delete method from the activity [note that the delete method means that the activity is removed from the PDPS database, as opposed to the destructor operation for the class, which deallocates the memory associated to the object within the application, the destructor is called after the delete].
4. The entry describing the plan itself in the database is deleted.

4.5.8.8 Event Trace

None.

4.5.9 Publishing a Plan Scenario

4.5.9.1 Abstract

This scenario describes the system response to a production planner publishing a plan which is maintained within the PDPS database. After being selected, the plan would be formatted by the system into a published plan consisting of three different types of file, which are Metadata file, ASCII report file, and Binary report file. The published plan then would be insert into Documentation Data Server. The method to insert a published plan into Document Data Server is an operation encapsulated in the PlPublishedPlan Class.

4.5.9.2 Interfaces With Other Subsystems and Segments

Document Data Server provides storage for published plans. The interface to the Document Data Server is shown by the DsDoProductionPlan object, which is Document Data Server abstraction for the plans.

4.5.9.3 Stimulus

A production planner initiates PublishPlan Function from the Production Planning Workbench.

4.5.9.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI

- PlPlan
- PlPublishedPlan
- PlPlanMetadataFile
- PlPlanASCIIReportFile
- PlPlanBinaryReportFile

4.5.9.5 Beginning System, Segment and Subsystem State(s)

Steady state, up and running.

4.5.9.6 Ending State

No change in the state of the system.

4.5.9.7 Scenario Description

1. The production planner starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17).
2. The production planner is presented with a function to publish a plan, and initiates the function.
3. Published Plan object is created.
4. The published plan is inserted into Documentation Data Server.
5. Planning System will send a notification by mail to a pre-prepared list of address when a new plan is published.

4.5.9.8 Event Trace

See Figure 4.5-11.

4.5.10 Building the Resource Configuration Scenario

4.5.10.1 Abstract

This scenario describes the initialization of the Resource Configuration from MSS

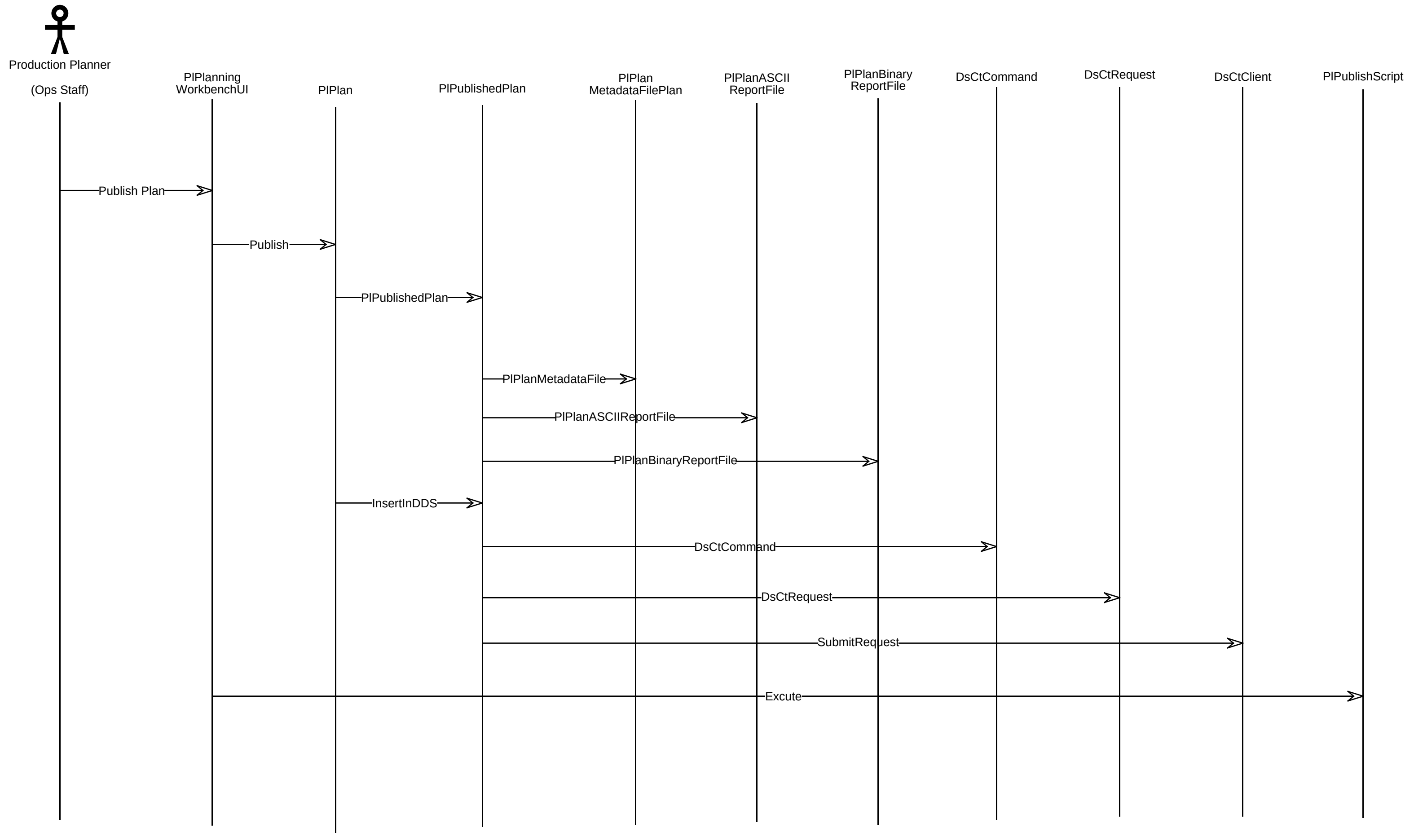
4.5.10.2 Interfaces With Other Subsystems and Segments

The MSS provides resource configuration information to the subsystems.

4.5.10.3 Stimulus

The resource configuration is built, or updated as a manual initiated operation from the Planning Workbench.

Figure 4.5-11. Plan Publications Even Trace



4.5.10.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
  PlResourceConfiguration
- PlResource
- PlString
- PlComputer
- PlDiskPartition
- MsDAAC

4.5.10.5 Beginning System, Segment and Subsystem State(s)

Steady state, up and running.

4.5.10.6 Ending State

No change in the state of the system.

4.5.10.7 Scenario Description

1.- The production planner starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17).

2. - The production planner is presented with a function to build the resource configuration, and initiates this function.

3. - MsDAAC object is created, and a filter applied to specify the need for information on production resources only.

4.- The iteration services of the MsDAAC class are used to extract the production resource and create the objects for these within the PDPS database.

4.5.10.8 Event Trace

See Figure 4.5-12.

4.5.11 Plan Activation Scenario

4.5.11.1 Abstract

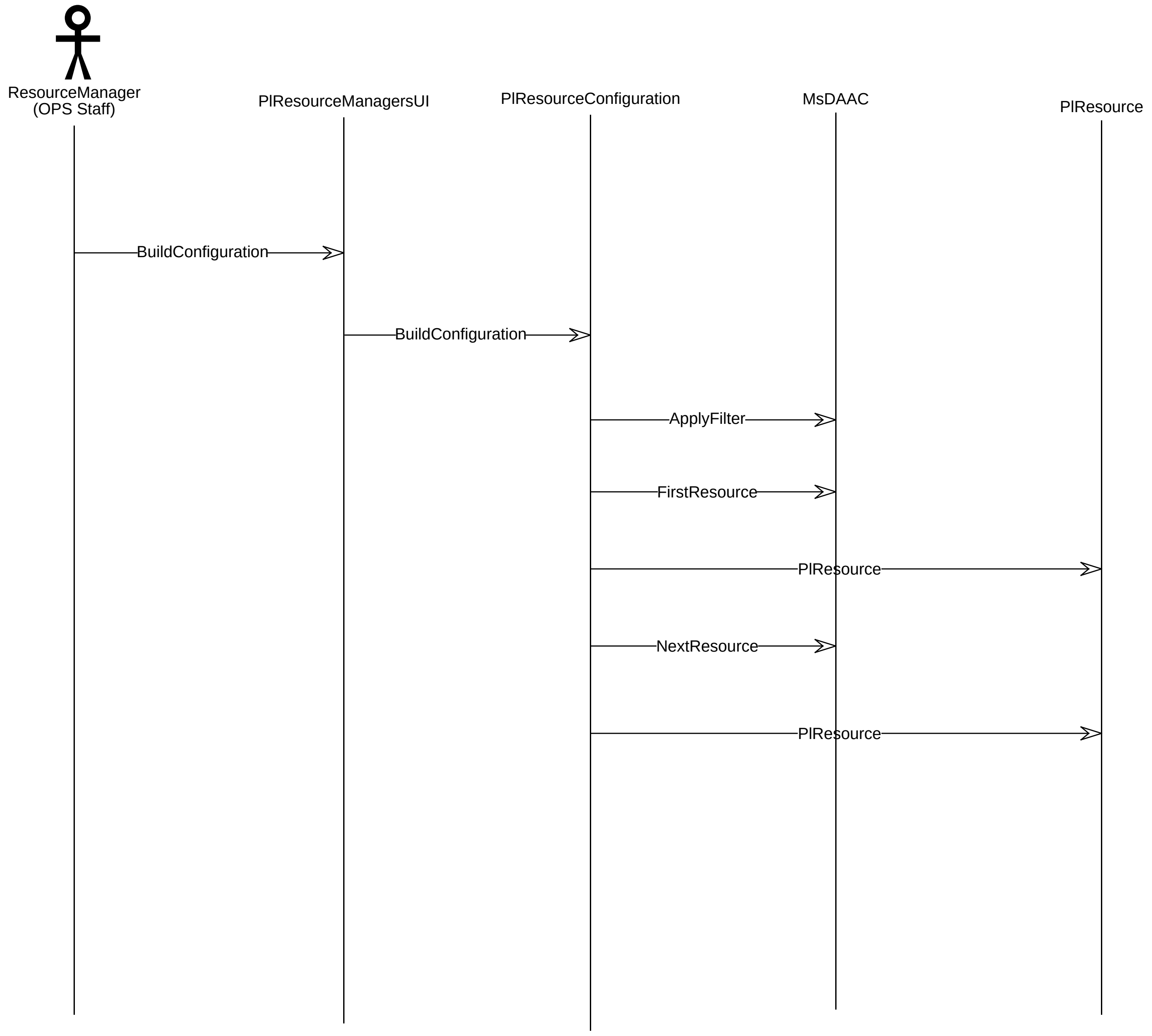
This scenario describes the activation of a plan from the Production Planning Workbench.

This scenario presents an abstract representation of the activities that occur within the Production Planning Object Library. For fuller description of the “planning/scheduling” aspects of this scenar io please refer to the Production Planning Object Library CSC section (section 4.6.5).

4.5.11.2 Interfaces With Other Subsystems and Segments

None.

Figure 4.5-12. Resource Configuration Event Trace



4.5.11.3 Stimulus

The production scheduler initiates the Production Planning Workbench in order to activate a plan.

4.5.11.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlPlan
- PlGroundActivies
- PlGroundEvent
- PlPGEActivities
- PlDPR
- PlActivities
- DPJobScheduler

4.5.11.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running. The scenario assumes that a previous portion of the plan has been activated (therefore there may be scheduled jobs within the data processing subsystem's job scheduler COTS).

4.5.11.6 Ending State

No PDPS database server state. A new portion of the plan is activated.

4.5.11.7 Scenario Description

1.- The user starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17). The user selects a plan which is to be used to drive the production schedule.

2.- On recreation from the PDPS database the plan is automatically updated to reflect changes that have occurred since the plan was generated, this includes updating the status of all the DPRs which were previously activated in the Data Processing Subsystem.

3.- The plan object determines whether any new ground events have been defined, or old ground events have been deleted since the plan was generated or last updated. The Activities describing the ground events within a plan are updated accordingly and the plan is updated.

4.- The plan object determines whether any of the data processing requests within the plan have been completed and updates the plan accordingly. The user may modify the plan at this point by redefining priorities in order to achieve a revised schedule, this thread is not described here.

5. The user initiates activation of the plan.

5.1 The plan object will create an ordered list of the activities which are currently activated in the Data Processing Subsystem.

5.2 A second list is created of all the activities within the scheduling window. This window defines the portion of the plan that needs to be rolled into the COTS. By default this window is 24 hours from the current time, however it is modifiable within from the workbench utility.

5.3 The lists of activities are compared. Any activity describing an active DPR which is not in the list within the scheduling window is canceled.

6.- The plan object iterates through the activities defined in the order indicated by the plan, and calls the appropriate scheduling operation for that activity.

6.1 If the DPR is already active within the Data Processing Subsystem then the update method is called in order to present a new predicted time of execution, and possibly a new priority.

6.2 If the DPR has not been previously activated then the schedule operation is invoked.

7.- The Data Processing Request activities will initiate a call to create a PGE job within the data processing subsystem's job scheduling COTS.

8.- The Ground Events will initiate a call to create a Ground Event Job within the data processing subsystem's job scheduling COTS.

Note: For details of how the jobs are created within the COTS and subsequently managed from the COTS please refer to scenarios within the Data Processing Design documentation.

4.5.11.8 Event Trace

See Figure 4.5-13.

4.5.12 Canceling a Plan Scenario

4.5.12.1 Abstract

This scenario describes the canceling of an active plan from the Production Planning Workbench. The scenario illustrates the interface to the Data Processing subsystem through the DPJobSched uler interface class.

4.5.12.2 Interfaces With Other Subsystems and Segments

The Planning subsystem interfaces with the Data Processing subsystem in order to cancel the scheduled Data Processing Requests.

4.5.12.3 Stimulus

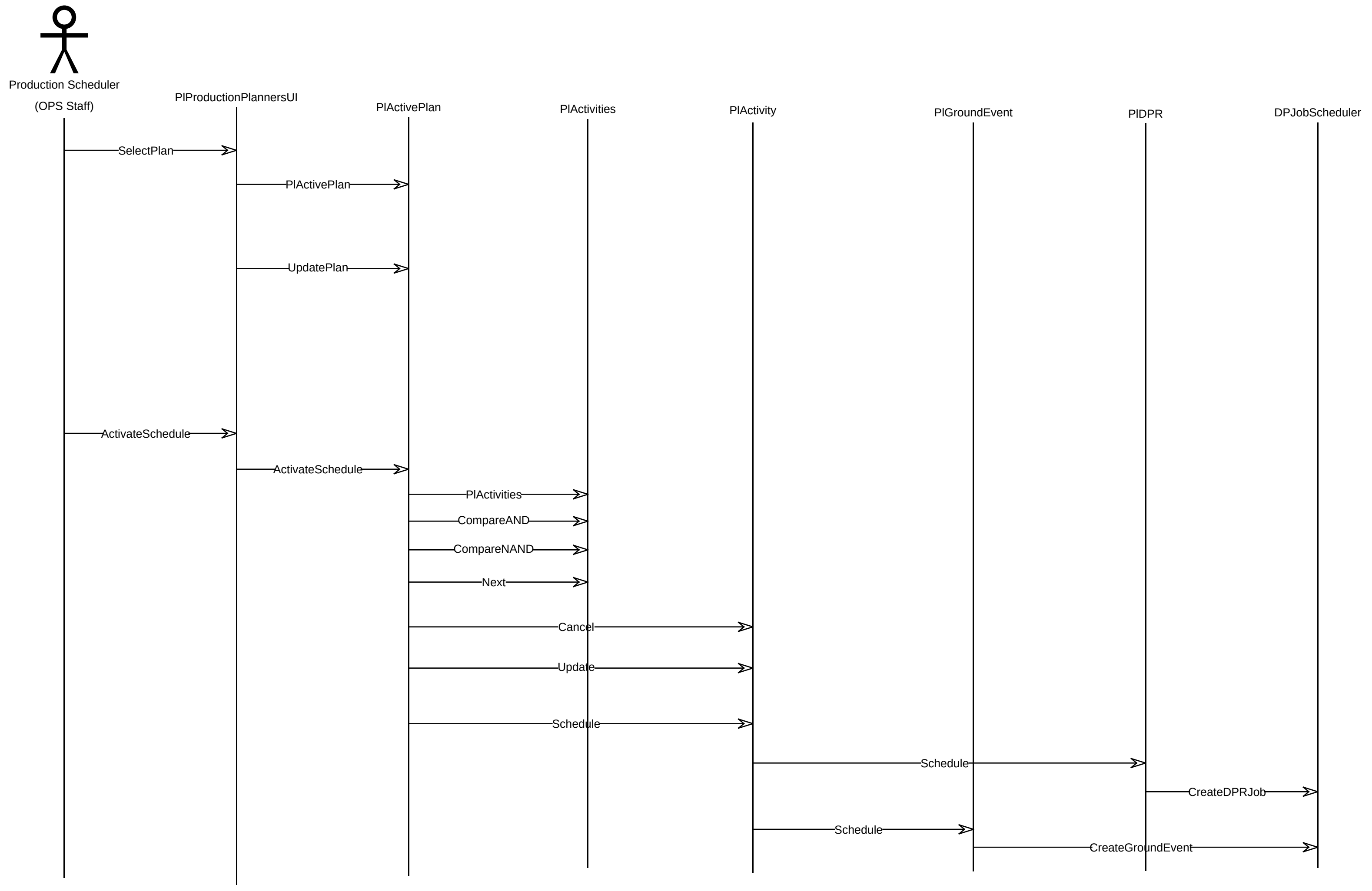
It's assumed that the Production Planning Workbench application is running and displaying the ac tive plan. The user decides to cancel the production schedule from the workbench.

4.5.12.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlActivePlan

Figure 4.5-13. Plan Activation Event Trace



- PlPGEActivities
- PlPGEActivity
- PlGroundEvent
- PlDPR
- DPJobScheduler

4.5.12.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running. A portion of a plan has been activated (there are scheduled jobs in the data processing subsystem's job scheduler COTS).

4.5.12.6 Ending State

No change in PDPS database server state. The scheduled activities are canceled.

4.5.12.7 Scenario Description

4. The operator initiates the command to cancel the production schedule.
5. The plan object creates an ordered list of the activities which are part of the active schedule (determined from the status attribute).
3. The plan object iterates through the activities and invokes the cancel method from the activity.
6. The PlDPR and PlGroundEvent classes interface with the DPJobScheduler class to cancel the activity.
7. The plan is updated.

4.5.12.8 Event Trace

See Figure 4.5-14.

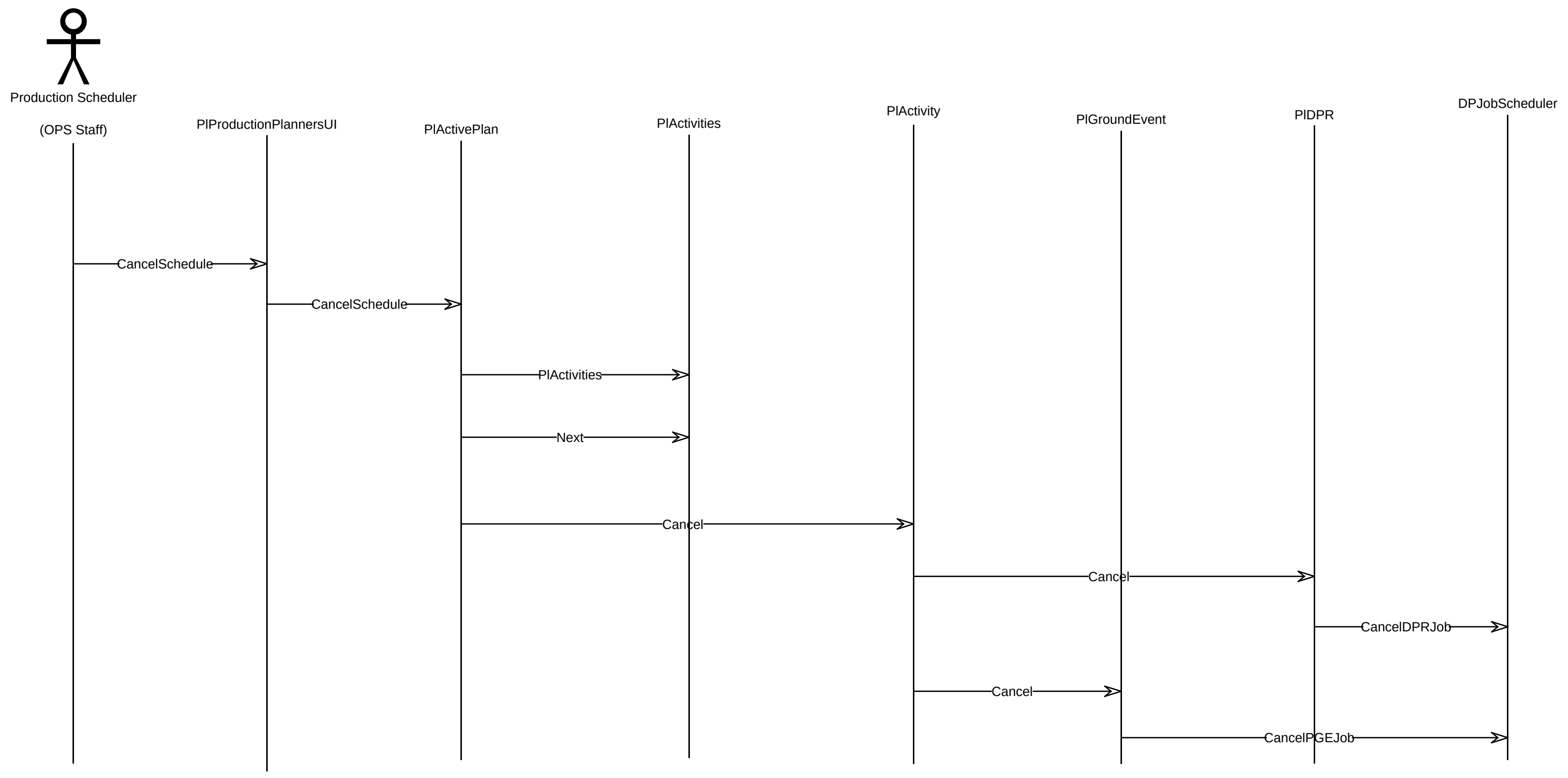
4.5.13 Statusing a Plan Scenario

4.5.13.1 Abstract

This scenario describes the statusing of a plan from the Production Planning Workbench. A com ponent of the Job Scheduling COTS, AutoXpert now provides the dynamic display of the produc tion schedule, therefore there are no requirements for dynamic updating to the plans. However there are occasions when the Plan from which the schedule originated would require updating, or statusing. Primarily the need for updating this will be before “downloading” a day's schedule into the COTS, taking into account the previous days production.

This scenario presents an abstract representation of the activities that occur within the Production Planning Object Library. The full detail of generating a plan is very complex, and intimately tied to the Production Planning Object Library. This scenario is presented to describe the process at a reasonable level of detail. For fuller description of the “planning/scheduling” aspects of this sce nario please refer to the Production Planning Object Library CSC section (section 4.6.5).

Figure 4.5-14. Plan Cancellation Event Trace



4.5.13.2 Interfaces With Other Subsystems and Segments

None.

4.5.13.3 Stimulus

The production scheduler initiates the Production Planning Workbench in order to activate a plan.

4.5.13.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlPlanningWorkbenchUI
- PlPlan
- PlGroundActivies
- PlGroundEvent
- PlPGEActivities
- PlDPR
- PlActivities
- DPJobScheduler

4.5.13.5 Beginning System, Segment and Subsystem State(s)

The PDPS database server is running. A portion of a plan has been activated (there are scheduled jobs in the data processing subsystem's job scheduler COTS).

4.5.13.6 Ending State

No change in PDPS database server state. The plan is updated.

4.5.13.7 Scenario Description

1.- The user starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17). The user selects the active plan which is to be updated.

2.- On recreation from the PDPS database the plan is automatically updated to reflect changes that have occurred since the plan was generated.

3.- The plan object determines whether any new ground events have been defined, or old ground events have been deleted since the plan was generated or last updated. The Activities describing the ground events within a plan are updated accordingly and the plan is updated.

4.- The plan object determines whether any of the data processing requests within the plan have been completed and updates the plan accordingly.

5.- The plan object creates an ordered list of the activities which are part of the active schedule (determined from the status attribute). For those activities which have not completed the plan object invokes the status method.

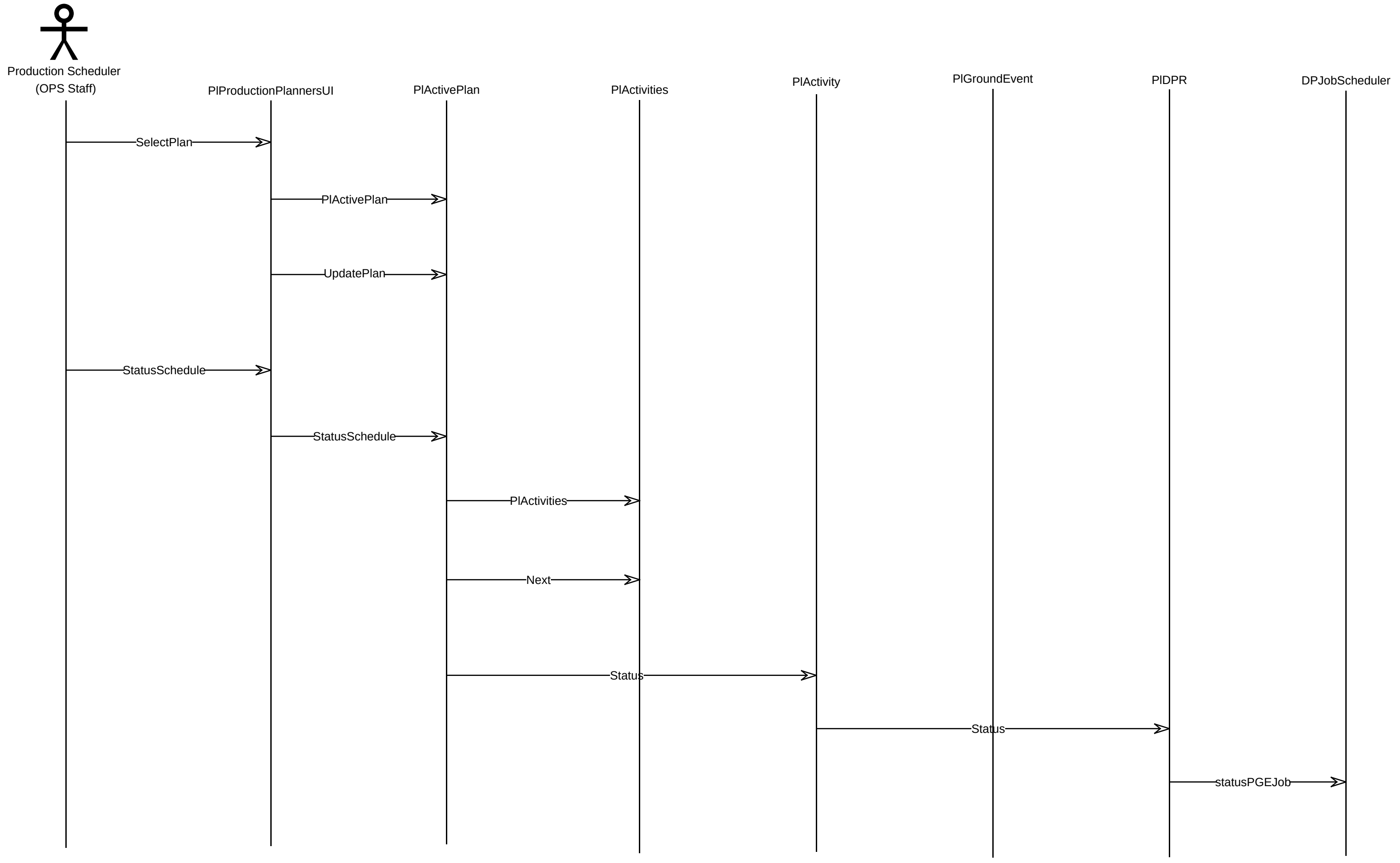
6.- The PlDPR and object interfaces with the DPJobScheduler class to return the status of the activity.

7. The plan is updated.

4.5.13.8 Event Trace

See Figure 4.5-15.

Figure 4.5-15. Plan Statusing Event Trace



4.6 CSCI Structure

The CSCI is decomposed into a number of CSCs. The CSCs correspond either to an application, or a class category describing a logically related set of functionality. The table below briefly out lines the CSC breakout of the PLANG CSCI.

Table 4.5-1. Production Planning CSCs

CSC	Description	Type	
Subscription Editor	Interfaces to Data Server for submission of Subscriptions required so that Planning subsystem is notified of arriving data.	DEV	
Production Request Editor	Accepts Production Requests which describe an order for Data production, translates to Data Processing Requests which specify the PGEs which have to be run to service the Production Request.	DEV	
Subscription Manager	Interfaces to Data Server for notification of data availability, releases Data Processing Requests queued in Data Processing subsystem.	DEV	
Planning Workbench	Provides the ability to create, modify, and activate a plan for the scheduling of Data Processing Requests.	DEV	
Planning Object Library	A set of C++ class libraries that provides a framework for the Planning Workbench.	OTS	DELPHI (Hughes class libraries)
PDPS DBMS	DBMS Provides persistent storage for Planning data (e.g. PGE Profiles, Production Requests, Data Processing Requests, Plans etc.)	OTS	Sybase

4.6.4 Production Planning Workbench CSC

4.6.4.1 Purpose and Description

The application is used to prepare a schedule for the production at a site, and forecast the start and completion times of the activities within the schedule. These functions provided by the workbench include the following high-level activities:

1)- Candidate Plan Creation -- from the production requests prepared by the Production request Editor

2) Plan Activation -- activating a candidate plan

3) Updating the Active Plan -- feedback from the processing into the active plan.

4) Canceling/Modifying the Active Plan

As described previously, activating a plan entails rolling a portion of a selected plan into the Au toSys COTS. This “schedule” is then managed within the Data Processing subsystem. The forecast times generated within the planner are used to set up operator alerts that would make the operator aware of departures from the predicted schedule. The production planning workbench can period ically update it's predictions using feedback from the AutoSys.

4.6.4.2 Mapping to objects implemented by this component

As described in sections 4.3.4 to 4.3.6.

4.6.4.3 Candidate products

The Production Planning Workbench is build largely from COTS components. The planning work bench is built on top of a Planning Object Library that provides a framework for the application. This object library is provided by C++ class libraries which are described in the following section.

Job scheduling COTS are coming close to providing more capabilities that would meet some of the production planning capabilities, none of the packages at present meet the full requirements of the ECS. The AutoSys / AutoXpert packages are developing capabilities that at later releases capabil ities may be suitable. The ECS is committed to influencing vendor direction to leverage capabili ties to cover additional of the ECS requirements.

4.6.5 Planning Object Library

4.6.5.1 Purpose and Description

The Planning Object Library provides a reuse framework for build the Planning Workbench appli cation. As described within the Planning Workbench CSC section 4.6.4, there is no complete “off the shelf” solution that covers all the Production Workbench requirements, and therefore reuse of robust class libraries is the best approach for developing this application. A number of planning and scheduling frameworks have been evaluated for scheduling in the Planning Object Library; these frameworks are described in the Scheduling Engine Evaluation Trade. In general, these li braries provide similar capabilities. The Hughes Delphi Scheduling Class Software has been se lected from those considered since it consists of generic, non-application specific libraries and there is a high degree of expertise within the ECS with using these Libraries, both within the Plan ning subsystem through prototypes and within FOS where these libraries are used in the mission planning.

4.6.5.2 Mapping to Objects Implemented by this component

The CSCI object model presented in section 4.3.4 showed an abstract view of the Production Plan ning Workbench. This was mainly due to the fact that the complexity of the planning and schedul ing aspects of the CSCI would overwhelm the rest of the model and scenario descriptions. The following sections first give an overall view of the Delphi scheduling class libraries and then show in detail how the Planning Workbench application is designed to reuse the Delphi libraries.

4.6.5.3 Hughes Delphi Scheduling Class Libraries

The Hughes Delphi Scheduling Software is designed to assist in developing efficient and effective scheduling and planning. It is composed of a set of user-oriented, integrated, modular tools.

The toolset provides building blocks to allow developers to:

_ Model system resources that reflect all relevant operating states and constraints.

_- Automatically create coarse or detailed schedules for all system resources based on requests, using a variety of tailorable scheduling algorithms.

_- Provide high functionality interfaces for planners to review and edit service requests, corresponding resource information, and generated schedules.

_ Provide support for interactive development of contingency or impact studies (*What-Ifs*).

Delphi is based upon a system of distributed, modular components. Each component represents a distinct planning function, and each component can be plugged in, disconnected, or replaced as changing concepts, system needs or software upgrades require. In addition, each component has been engineered utilizing object-oriented methodologies. This provides many significant benefits, including:

_ Functionality - The software can be easily tailored to implement application specifics.

_- Extensibility - Once delivered the system can be easily enhanced without the need for complete replacement or extensive and costly block changes.

_- Maintainability - All functions (data and process behavior) are well encapsulated in the software architecture. Therefore, should a problem occur it can be easily identified and isolated. Metrics for maintenance indicate exceptional savings for customers in O&M costs for this software.

These features combine to support a high degree of system flexibility and expandability, and in the long term contribute to a significantly lower system life-cycle cost.

All of the tools in Delphi are built on a foundation called the Hughes Class Library (HCL). This product provides a framework for all the objects in the system and provides generic functions for services such as time, collections (list, sets, arrays, etc.), stream input and output, inter-process communication, and windows displays.

4.6.5.3.1 Delphi Resource Model

At the heart of Delphi is the Resource Model. In order to generate a timeline schedule for activities the Planning Workbench system must have detailed knowledge of both the activities that are to be performed and the resources that are required to be utilized or expended in order to complete each activity. Delphi's Resource Model supplies the structure to define these entities. In addition, the Re source Model acts as the owner of all data used by Delphi and therefore provides all data manage ment services to the toolkit and the user.

The Resource Model consists of resources, resource states, and all relevant resource constraints.

Resources. All resources that are necessary for planning and scheduling are implemented as ob jects within the Resource Model. Application specific resource classes/objects can be derived from the resource classes provided in Delphi. These more specific classes/objects contain the attributes and behaviors that are unique to them.

Resource State. Besides modeling real world objects, another function of the Resource Model is to keep track of the state of all resources over time. Real-time and periodic (batch) updates of the state of all components of the system are sent to and stored by the resource model. For example, the state of a resource may indicate whether it is available or unavailable, and the nature of its current tasking.

Activities. In addition to resources and resource states, the resource model contains activities. These are schedulable entities that represent system tasking. During scheduling, resources are as signed to an activity. The resource's state is then updated to include each new activity. In turn, these updates can be directed as real-time modification to the timeline.

Constraints. Also in the Resource Model are the constraints present in the planning system. The scheduling algorithm will consider constraints between resources when attempting to properly al locate resources in a given plan.

The Resource Model defines, in both data structure and functional behavior, the resources being utilized. Any constraints concerning reasonable, proper, of safe behavior are defined to the system. The Resource Model also retains control over the definition of activities. These activities define a sequence of operations that are required to perform a high level goal. Both elements to scheduling, requests and resources, are managed within the flexible architecture of the Resource Model. Using these pools of information, the scheduling algorithms optimize the application of activities to re sources to develop constraint-free plans for operational use.

Resource Model Hierarchies

There are two primary hierarchies within the Resource Model: whole/part activities and resource activities. The whole/part hierarchy provides an aggregation of many associated system objects; for instance, a “whole,” such as the plan, contains “parts,” such as resources and activities. In this manner. the Resource Model relates its individual inheritance hierarchies to create a global, con solidated, constraint-free plan.

A second hierarchy involves the individual resources and activities with which the planning and scheduling system must deal. In general, a resource hierarchy provides a detailed model of all rel evant system resources. The activities are defined as resource-independent descriptions of opera tions that ultimately will be assigned to resources at specific times.

Each resource is responsible for maintaining a record of its state through time, for exporting algo rithms of general interest, and for modeling and applying constraints. The resource state models what the resource is doing through time and may be different for each kind of resource. Resources know what plans are available and can have a different state on different plans.

Use of Resource State

Clients of the resource model can interact with resource state in several ways. A client can interact with resource state directly by asking a resource for all state on a given plan for a given time inter val. The client would then have a detailed knowledge of the resource and can have as much knowl edge of the resource and its inner workings as appropriate.

A client can also ask a resource for time intervals in which particular constraints are satisfied. These intervals may be displayed on another process, for example a timeline, in order to commu nicate to the user the window of opportunity for the resource. To obtain these intervals, the resource would iterate through all of its states and check with compatibility against existing state.

Resources can have a heterogeneous state. For examples, a resource could be tasked for a time in terval and also be unavailable for another time interval. Both of these cases are supported by the resource state mechanisms.

Scheduling Activities to Resources

Resources that can be tasked can generally be asked to allocate (check constraints), unallocate (re move tasking), force allocate (do not check constraints), check allocate (check constraints but do not change state), and when allocate (tell me when constraints, if ever, are satisfied for a given set of tasking). This functionality is generally used by schedulers. For example, a sequential scheduler could order activities and then try activity/resource combinations until the activity was scheduled or until no combinations were left. The sequential scheduler will then move on to the next activity. Activities can be ordered by user-set priority, laxity, availability, or any other mechanism. The im portant point is that the scheduler only organizes the activities and orders the requests for resources. The resources themselves know what constraints to check and how to generate resource state. The scheduler may have to have resource-specific knowledge during the ordering process, or may use heuristics to optimize utilization of a resource.

4.4.5.3.2 Delphi Architecture

The following section describes the software architecture used by Delphi. Delphi is composed of a series of modular *libraries* of software. Each library defines a series of groupings of both data and functions that are referred to as classes. The classes serve to provide the mechanism for data structures and manipulation functions to be encapsulated or localized, thereby providing discrete functionality that can easily be developed and debugged. These objects provide the atomic func tionality to all Delphi tools.

4.6.5.3.3 Hughes Class Library

The Hughes Class Libraries form the core from which all scheduling products are developed, thus providing real cost savings in development, test time, and maintenance costs. The Hughes Class Libraries are implemented in C++ according to the current standard, as set forth in the AT&T Ver sion 2.0 C++ Programming Language Standard. The bottom layer of software usage is composed of X-Windows code, per the X11 standard. HCL utilizes X11 and allows the user application code to make direct calls to X11, if necessary. Within HCL is a display class of utilities that support the Motif display standards. These libraries have been in existence since 1990, and have been thor oughly tested and fielded in several operational systems.

The Hughes Class Library (HCL) is a library of C++ class declarations. These declarations are gen eral purpose programming utilities, and include:

_ display classes (XView, XGL, and Motif)

_ collection classes

_ Inter-process communication classes

_ other miscellaneous utilities (e.g. string, rectangle, command line, etc. classes)

HCL contains several libraries: misc, dispx, mdisp, and ipc. These are summarized below:

misc Library

The 'misc' library supports the collections in linked list array, or set format. It provides iterators over each type of collection for ease of movement throughout the collection. Collections can safely have multiple concurrent consumers. HCL allows classes derived from a common base class (HObject) to be stored in the same collection, thus allowing heterogeneous collections.

The choice of collection type (array, list, set) does not affect the application code. These collection types all are derived from a common class that established the protocol for the derived classes.

This library also contains classes that provide various date/time functions, command line informa tion, timer functions, string functions, and vector and matrix functions.

dispx Library

The 'disp' library includes classes for drawing using XGL utilities. It has classes that encapsulate XGL contexts, rasters, fonts, and color usage. It also has classes to maintain display regions and collections of sub-regions.

mdisp Library

The 'mdisp' library supports displays with the look and feel specified in the Motif Style Guide. The library uses the Motif toolkit from the Open Software Foundation (OSF). The Motif toolkit is based on the X-Toolkit Intrinsics (Xt), which is the standard mechanism on which many of the toolkits written for the X-Windows System are based. The user will notice that the library encapsulates ca pabilities at all three levels (i.e. Motif, Xt, and Xlib).

The library provides classes that allow the user to create and manipulate X-Windows, color maps, events (i.e. keyboard, mouse buttons, window enter/exit), and user interface objects called widgets (i.e. menu bars, pulldown menus, buttons, scrollbars). There are classes that provide displayable regions, subregions, rectangles, scalable fonts, strings, and colors. Mdisp also provides a class im plementation of the graphics context (GC).

The DAppl class provides a template for a display application. It provides the behaviors to create, run, and destroy the application. The class does not provide a base frame.

ipc Library

The 'ipc' library contains classes that encapsulate interprocess communications, providing a simple interface for the programmer. It is an implementation of Berkeley sockets and XDR streams.

Messages derived from a common message class (HMessage) can be passed between processes without the ipc code knowing anything about the contents of the message. This isolates the code that does know about the message content, making for a much easier, faster development and easier maintenance.

4.6.5.3.4 Delphi Reuse Libraries

The framework for all scheduling applications is HCL. In addition to this core product are reusable libraries of classes for virtually all aspects of a scheduling system. For purposes of discussion, these libraries will be grouped into four categories. These categories are: Scheduling, Resource, Timeline, and Planning Class Libraries. All libraries build upon the base of the HCL and are de veloped in the C++ programming language.

4.6.5.3.4.1 *Scheduling Class Library*

The Scheduling Class Library provides a framework for the incorporation of scheduling algo rithms.

Reusable Classes:

SResource (class modeling resources with scheduling) - Each SResource specifies protocol for allocation of an activity to a resource for a time interval.

SRsPool (scheduling resource pool) - An instance of this class provides a storage mechanism for quick retrieval of scheduling resources (indexing by associated resource ID). Derived classes can provide special queries that are typically process non-specific. For example, fill a given collection with all scheduling resources of a given type that have no state for a given time interval.

SSimpRs (class modeling resources with mutual exclusion) - Each SSimpRs is associated with an RResource and implements allocation members that assume a mutual exclusion constraint.

SUpdCatAbs (resource model change notifier) - SUpdCatAbs is a base class that sets up protocol for registration of changes of resources, their states, activity changes, and plan changes. It also has protocol for flushing these change notices. Derived instances of this class might keep lists of clients interested in changes, and notify these clients when changes take place.

SaActPriSrtr (activity priority sorter) - This class customizes the sorter for activity priority.SaAllImpct (Impact Scheduler) - Instances of SaAllImpct know how to accept lists of activities and edit the resource model accordingly, using impact scheduling. The plan that is edited is the one already set by using the SaComponent member function plan().

SaAllNImpct (Non-Impact Scheduler) - Instances of SaAllNImpct know how to accept lists of activities and edit the resource model accordingly, using non-impact scheduling. The plan that is edited is the one already set by using the SaComponent member function plan().

SaAllocator (activity allocator) - Abstract class used to define generic protocol for allocating schedulers.

SaArySorter (array sorter) - This class uses the system sort algorithm to perform sorting.

SaComponent (scheduling component) - Abstract class used to define generic protocol for scheduling algorithm components (sorters, filters, allocators, refiners and post processors).

SaFastFit (fast fit allocator) - This schedule establishes protocol for fast fit scheduling. The algorithm used is a first come, first serve algorithm.

SaFilter (allocation filter) - Abstract class used to define generic protocol for filtering an area of a plan.SaInflator (allocation inflation post processing) - This scheduling component class tries to inflate allocations.

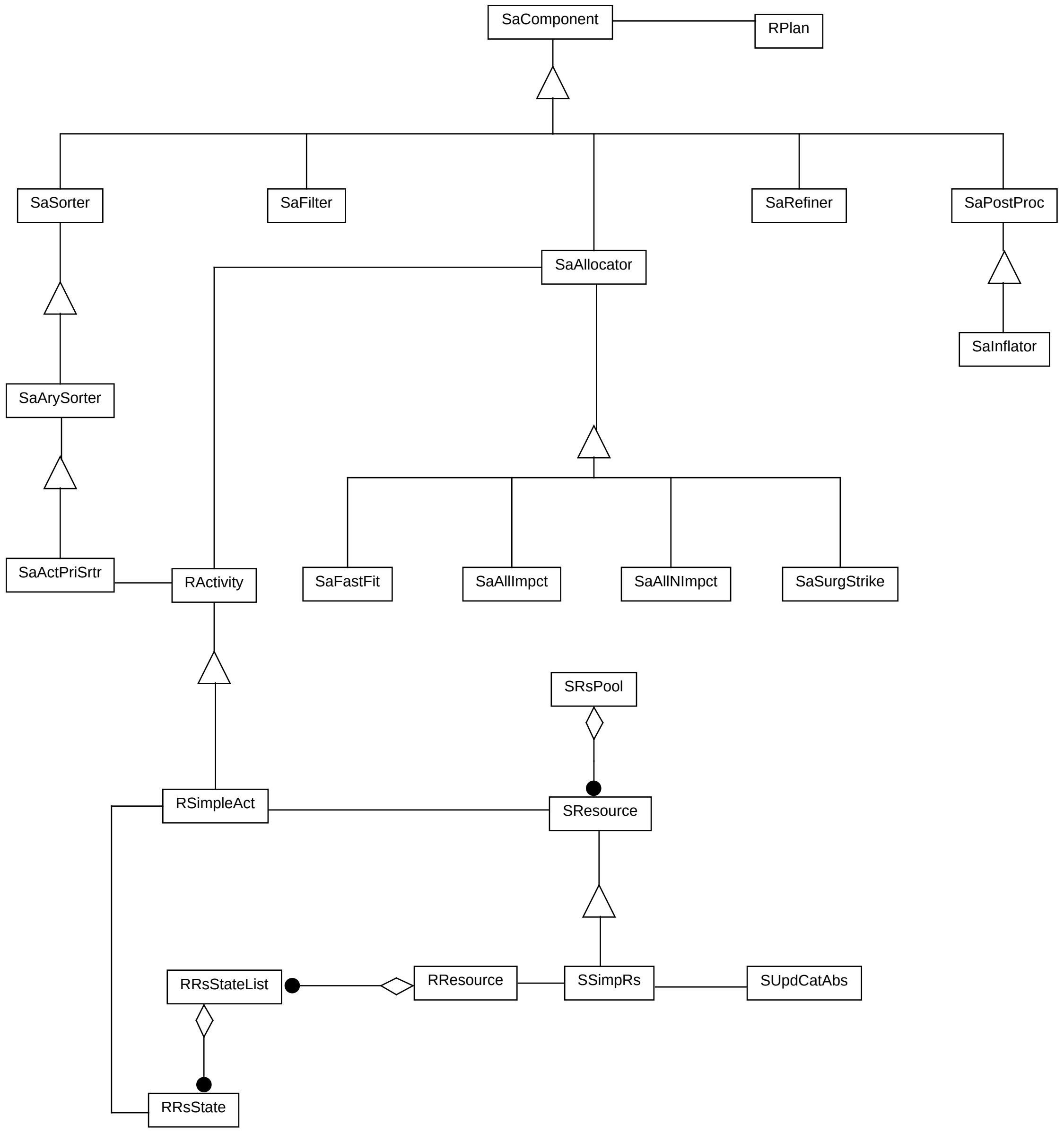
SaPostProc (allocation post processor) - Abstract class used to define generic protocol for post processing schedulers.

SaRefiner (allocation refiner) - Abstract class used to define generic protocol for refining an area of a plan.

SaSorter (activity sorter) - Abstract class used to define generic protocol for sorting a collection of activities.

SaSurgStrike (surgical strike allocator) - This schedule establishes protocol for surgical strike fit scheduling. The consumer can provide an actual allocation algorithm.

Figure 4.6-1. Delphi Scheduler Model



Customization of the Scheduling Class Hierarchy:

The Delphi Scheduling Class Library provides a framework for incorporating a variety of algorithms in the Planning Workbench. The Planning Workbench reuses the classes the Scheduling which define protocols for sorting, filtering, resource activity generation, and refinement. It is the goal of the Planning Workbench to employ the Scheduling framework to construct algorithms which optimally allocate activities to resources, especially local disk storage.

4.6.5.3.4.2 ***Resource Class Library***

The Resource Class Library provides the structure for defining application domain resources and it builds upon both HCL and the Scheduling Class Library. The Resource Class Library provides templates to define resource models capable of managing discrete resource states and mechanisms for assigning resource availabilities to resources. In addition, inter-request correlations, such as pre-requisite, co-requisite, and post-requisite constraints, preferences, consumable resource modeling, and specific configuration requirements, are addressed. Each of these resource definitions will be used by the schedule deconfliction processes to provide a constraint free plan. When an activity is defined, multiple resources can be identified as being applicable to that activity.

Reusable Classes:

RActAll (information about an activity allocation) - This class adds an activity to the base allocation. An example use is as a base class for modeling allocation of an activity to a bunch of resources. In that case, the derived class would have explicit resource support.

RActIdFactAbs (Unique activity id factory) - Instances of RActIdFactAbs are activity id generators. You can ask them for the next allowable activity id that can be assigned. This class has the notion of a global RActIdFactAbs object and provides a static nextId member function so that you do not have to have a specific instance everywhere you want to use it.

RActIdFactMem (Unique activity id factory) - Instances of RActIdFactMem are activity ID generators. You can ask them for the next allowable activity ID that can be assigned. This class initializes its activity ID range from the current activity pool.

RActPool (activity pool) - An instance of this class provides a storage mechanism for quick retrieval of activities (indexing by activity ID). Derived classes can provide special queries that are typically process non-specific. For example, fill this collection with all activities of a given type that have been allocated on a given plan.

RActState (tasking on a resource) - RActState is a resource state class that is generated by an activity.

RActivity (scheduling activity) - This is a base class for scheduling activities. It establishes protocol for all derived activities.

RAll (information about an allocation) - Each allocation has a time interval an optional plan name and a lock. If the plan name is empty, the allocation can pertain to all plans.RComplexAct (collection of activities) - Each instance of this class holds a collection of activities.

RDegState (tasking on a resource) - RDegState is a resource state class that represents a degraded resource. Presence of a direct instance of RDegState means that a resource is broken. Derived classes can represent how a resource is degraded.

RPlan (plan) - This is a base class that models the allocations of plans to resources.

RPlanPool (plan pool) - An instance of this class provides a storage mechanism for retrieval of plans (indexing by plan name). Derived classes can provide special queries that are typically process non-specific. For example, fill a given collection with all of the activities in the activity pool that have different allocations on two plans.

RResource (class modeling an entity with state through time) - RResource is a class that models an entity that has state through time. State is kept in RRsStateLists. Each RResource may have more than one RRsStateList. These state lists contain information which is mapped to a particular name (a “plan”). State lists usually contain things derived from RRsState. In other words, each resource object may maintain state information for multiple plans simultaneously. There is an additional, plan-independent state list that each RResource maintains. This state list may be used to hold plan independent information such as state that applies across all plans. RResource objects maintain a notion of a “current” state list.

RRsPool (resource pool) - An instance of this class provides a storage mechanism for quick retrieval of resources (indexing by resource ID). Derived classes can provide special queries that are typically process non-specific. For example, fill a given collection with all resources of a given type that have no state for a given time interval.

RRsState (base class for resource state) - Abstract class used to establish protocol for any kind of resource state.

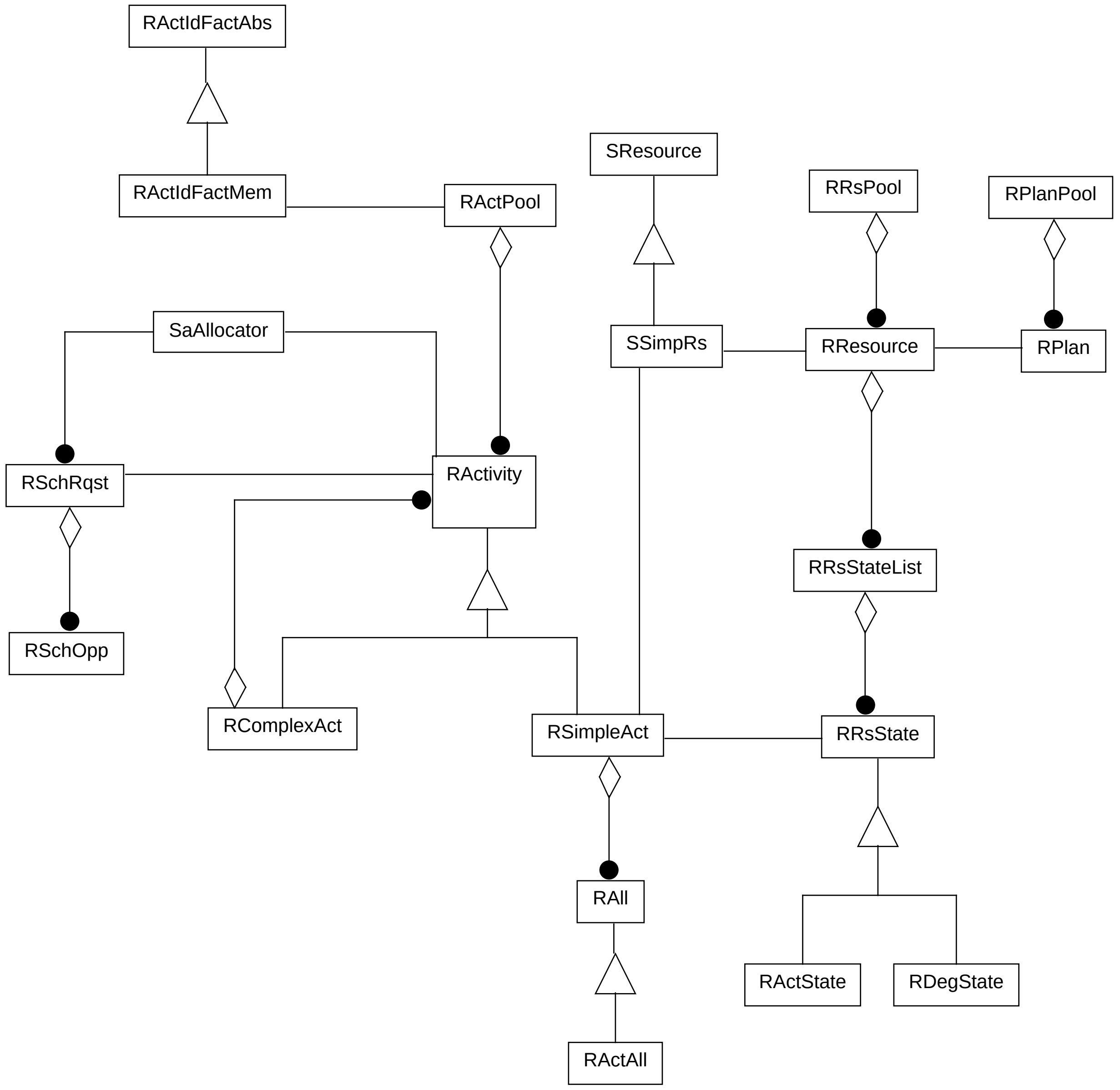
RRsStateList (resource state list) - An HEpochIntervalList with a name. An important difference between this class and an HEpochIntervalList is that this class empties() its contents upon destruction (unlike a vanilla HObjCollection, which clears() it's contents).

RSchOpp (a scheduling opportunity) - Each scheduling opportunity has a time interval and a resource id, representing an possible allocation of an activity to that resource and interval.

RSchRqst (resource scheduling request) - This class is a list of scheduling opportunities, which are specified as instances of RSchOpp, with an activity id specifying the activity to be allocated using the opportunities and a parameter indicating how many of the opportunities must be satisfied for the entire scheduling request to be satisfied. Only derivations of RSchOpp should be added to this class, and the add behavior of HObjList has been overridden to enforce this.

RSimpleAct (simple scheduling activity) - Each instance of this class models something happening on a resource in the system. Simple activities add allocations to the concept of Activity.

Figure 4.6-2. Delphi Resource Model



Customization of the Resource Model Hierarchy:

The Planning design takes advantage of Delphi by utilizing the object-oriented mechanisms of inheritance. Elements of the Resource Model hierarchy have been customized for the Planning problem domain. Derived resources have been designed to customize resource state and override or add specialized algorithms and constraints. Some derived resources add scheduling operations which take activities and check constraints and generate resource state through time.

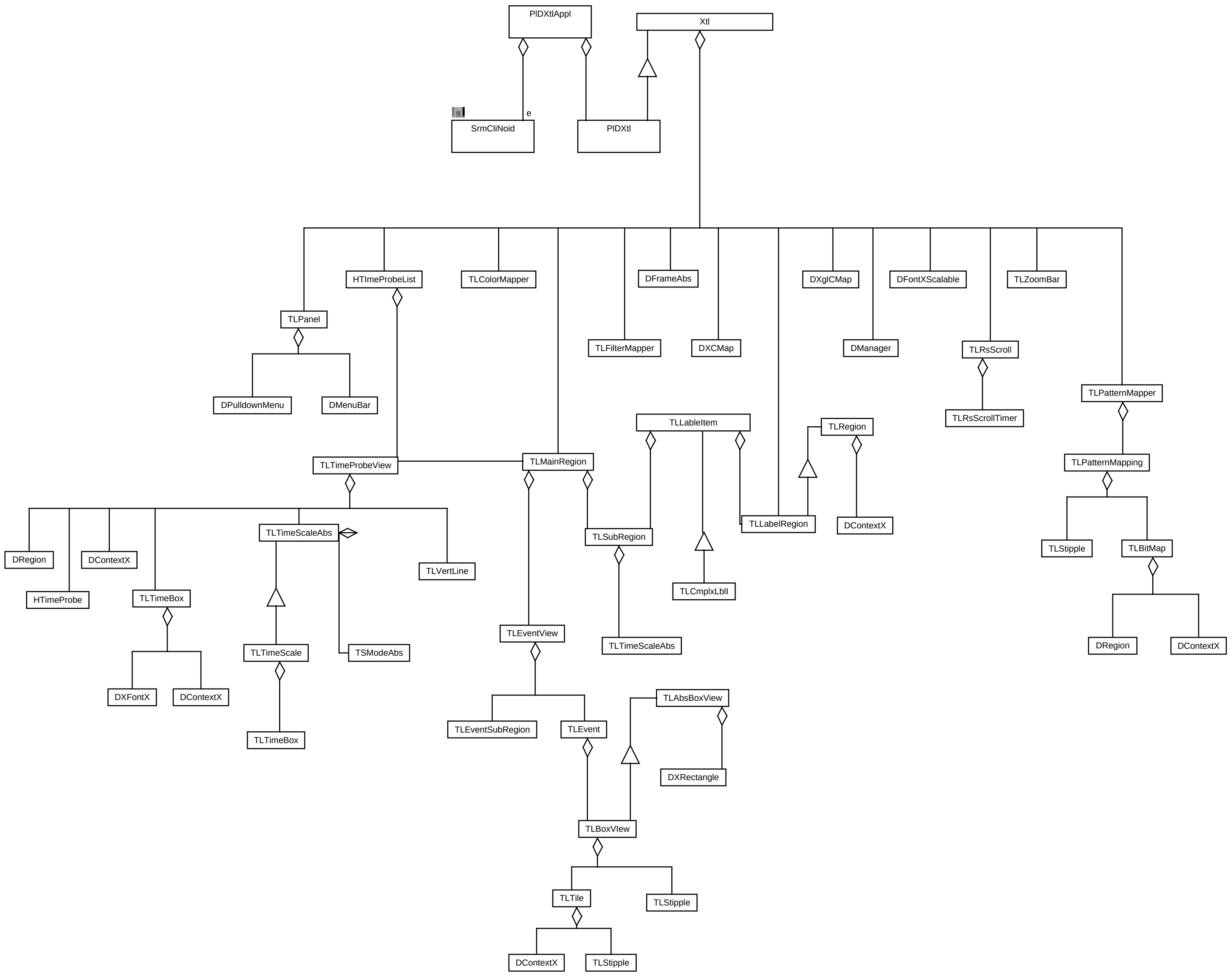
The derived Planning resources have been designed in a parallel hierarchy, as a result keeping the natural resource hierarchy and the scheduling hierarchy separate. Derived activity classes have been created that model the types of activities appropriate for the Planning resource model. A key point is that the specific additions to Planning system have been built upon the foundation already provided by the existing Delphi Resource Model structure. Many additions are simple derivations of existing planning and scheduling objects.

4.6.5.3.4.3 *Timeline Class Library*

The Timeline Class Library provides mechanisms to support the displays of time-ordered information in a graphical manner. The generic Delphi timeline contains a two-dimensional, composite region displaying resources and their use versus time. Each display is composed of rectangular subregions that represent resources aligned down the vertical axis to be viewed over time, and time across the horizontal axis. Scrollable windows are utilized to allow the user to manipulate the displays to view the required resource profiles. Events for any period of time are modeled and displayed. Each event is indicated on the display with a text-label, color and state information, all of which can be customized for the specific application. The Timeline is a powerful visualization device allowing the user to navigate hundreds of resources profiles. Each event on the timeline canbe compressed, expanded, edited, unallocated from its resource, locked and unlocked. Visually the user sees more detailed information the shorter the timespan being viewed, thus allowing a zoom cycle capability for all Timeline displays. Use of interactive point-and-click devices speed the user through birds-eye views of schedules down to the component parts and resources allocated to a single user request on the timeline. The timespan is adjustable and has no technical limits (although practical limits may insist only a portion of large schedules be viewed at any time).

The Timeline Class Library provides class implementations for drag-and-drop features between subregions, event manipulation, and supports multiple views of events. The use of color to convey importance, hierarchy, or groupings can be defined by the Customer, and color changing functions are provided with the Timeline classes. Scroll bars and multiple windows are supported. A time probe that anchors the center of focus of the display and provides default values of the time window of interest to other Delphi components, such as schedule Activity Editors, is also provided. The Timeline classes establish a generic structure for events.

Figure 4.6-3. Delphi Timeline Model



Customization of the Timeline Class Hierarchy:

The Timeline is one of the primary visualization tools within the Planning Workbench. The Timeline will provide a GANTT style view of the requested allocation of PGEs. In the Timeline, horizontal boxes will represent PGEs, and the boundaries of the boxes will represent start and end times. The Timeline enables the user to view in an instant the breadth of system resource utilization over time. Information displayed on the box (textual, color, graphical, etc.) will aid the user to search for individual PGEs. Several instances of the Timeline can be viewed simultaneously to compare the results of various scheduling algorithms.

4.6.5.3.4.4 ***Planning Class Library***

The Planning Class Library provides the display and integration structure to all Delphi scheduling applications. This library provides the interface slots to allow a developer to build up from the foundation of the components of Delphi into a coordinated scheduling environment. Additional capabilities provided within the Planning Class Library are: error message display and logging, distribution of resource states and coordination of schedule modifications, resource editing, scheduling activity creation, deletion, and dissemination, (planning synchronization across multiple users).

4.6.5.4 Use Of Delphi In The Planning Workbench

4.6.5.4.1 Planning Workbench Architecture

Delphi takes advantage of a client server model to allocate various display and algorithmic functions to separate physical processes. As applied to the Planning Workbench, this model is illustrated in Figure 4.6.4 below.

At the heart of Delphi and thus the Planning Workbench is the System Resource Model. This is the key server process in the Delphi client server model. All interfaces to persistent data required for the Planning Workbench are maintained within the Resource Model. The persistent data items relevant to the Planning Workbench are the Plans, Resources and Activities. The persistent data is maintained in the PDPS database. The Resource Model retrieves data from the database to describe the Resources and Activities and previously generated Plans, the Resource model stores data in the database to describe new or modified Plans.

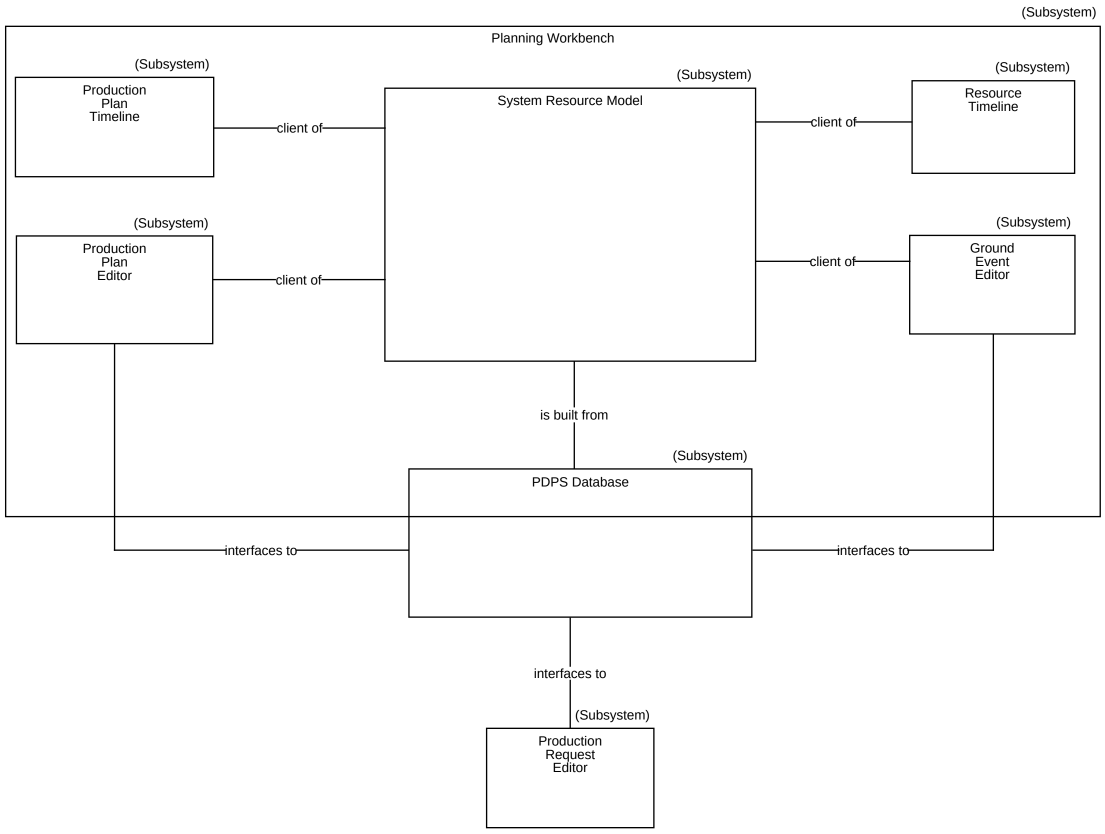
The clients of the System Resource Model are also shown in Figure 4.6-4. The Production Plan Editor is the key client within the Planning subsystem, this contains the planning algorithm responsible for generating the plan. The Planning algorithm is implemented within the framework of the Delphi System Scheduling Interface which encapsulates the protocol for planning algorithms.

The ground event editor is a relatively simple application which permits a resource planner (OPS role) to enter ground events that are stored in the PDPS database and displayed on the Resource Timeline.

Two timeline displays are shown in Figure 4.6-4 to represent the two basic views available for the operators, a resource ground event view, and a production plan view. The application that provides these views will be the same application, but with different filters applied to plan information.

To complete the picture, Figure 4.6-4 also shows the Production Request Editor interfacing to the PDPS database. The production request editor is the originator of the Data Processing Requests objects within the database from which the planned activities are created.

Figure 4.6-4. Planning Workbench Architecture



The main areas of the Planning Workbench high level model that were presented as abstractions in section 4.3 and are as follows:

PlResource:	This class maps to a number of classes which are specializations of the Delphi classes. The PlDRResource class and PlDSResource class special ize from the RRsource SResource classes respectively. These classes are used within the Delphi System Resource Model as described in section 4.6.5.4.2
PlPlan:	This class maps to the PlDRPlan class, which is a specialization of the RPlan class within Delphi. The RPlan class contains basic operations to manage and copy plans, these are some of the key features that are imple mented in Delphi which give considerable reuse advantage to the ECS Planning Subsystem.
PlActivity:	This class maps to the PlDRActivity, PlDRSimpleAct classes, specializa tions of the RActivity, RSimpleAct classes within Delphi.
PlResourceManager:	This class again maps to a number of classes which are specializations of Delphi classes. The Planning subsystem specializations are PlDSsiSched uler, PlDSAllocator, PlDSsiOppGen these classes are the key classes in implementing the Delphi System Scheduling Interface as described in section 4.6.5.4.3

There are in addition a number of the utility classes for collections, lists etc. that are provided by Delphi, and the lower level HCL class libraries which will be reused and specialized within the Planning Workbench application.

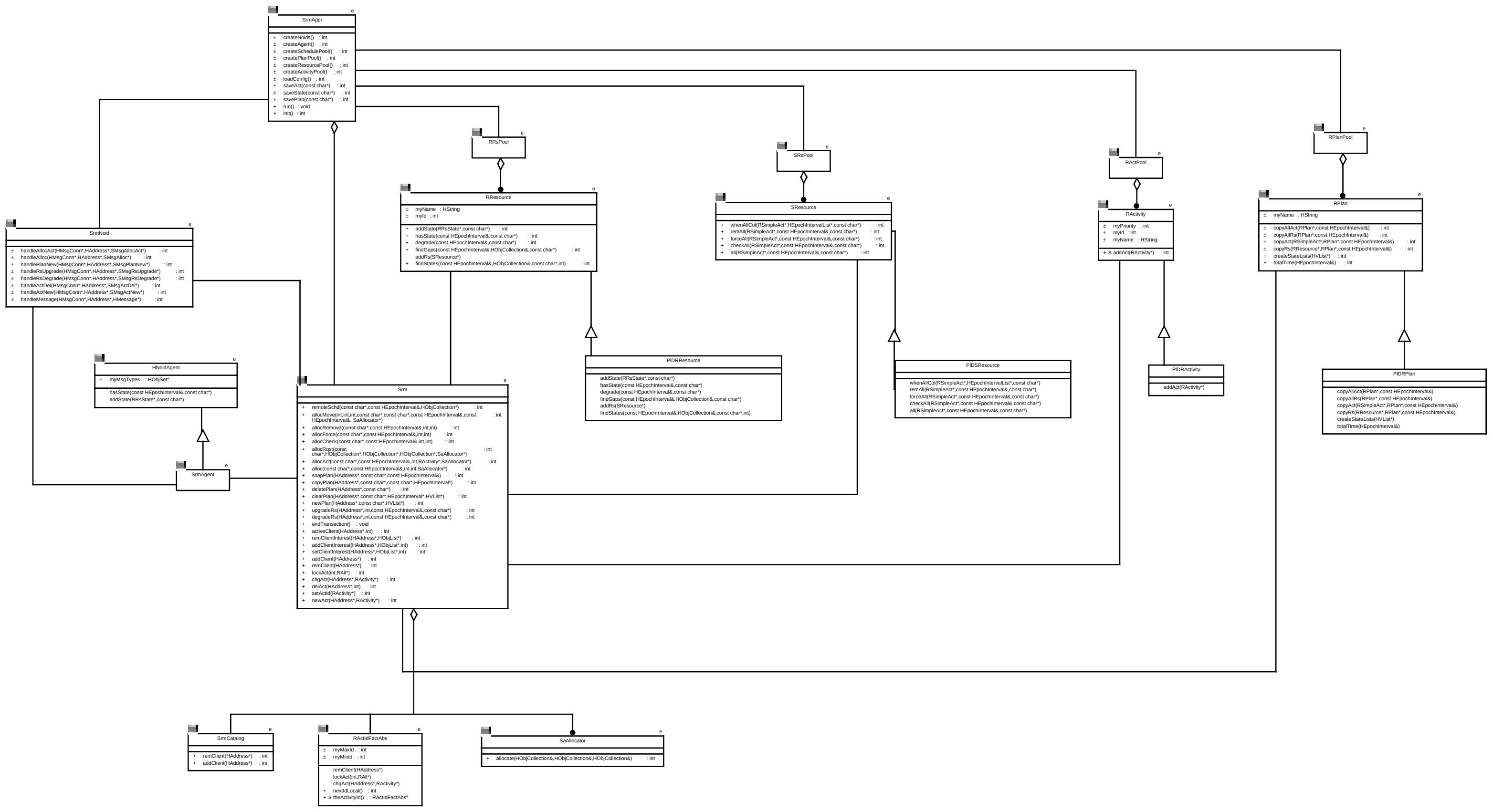
In order to give a more complete picture of these methods and to illustrate the Delphi reuse this is described in more detail here, first by describing the System Resource Model, then the System Scheduling Interface within the Production Plan Editor responsible for performing the planning al gorithm.

4.6.5.4.2 Planning Workbench System Resource Model

In order to describe the Delphi scheduling approach the System Resource Model should first be described. The object model diagram for the System Resource Model is shown below.

The System Resource Model Srm class is responsible for maintaining the resource model for the Planning subsystem. It controls additions, deletions, and updates to the resource model and man ages distribution of model changes to interested clients. When the Srm is initialized, all required pools are created: the resource pool, the scheduling resource pool, the activity pool, and the plan pool. The Srm then waits in the background for client messages which contain requests for infor mation about or updates to the resource model pools.

Figure 4.6-5. Planning Workbench System resouce Model



4.6.5.4.3 Planning Workbench System Scheduling Interface

This model describes the key portion of the Production Plan Editor within the Planning workbench to describe the plan generation and resource allocation scheme. The System Scheduling Interface (SSI) tool provides an application framework for performing generic sheduling. The SSI contains a user interface which allows the user to select activities to schedule for a chosen plan. The SSI communicates with the System Resource Model (SRM) to request the current state of the resource model. The SSI maintains copies of the SRM pools for efficient local scheduling. After activities are scheduled on local copies of resources, messages are sent to the SRM to update the central resource model.

A generic scenario that shows how activities are allocated to resources is given below.

4.6.5.4.4 Scheduling an Activity

4.6.5.4.4.1 Abstract

This scenario describes the system response to a production planner scheduling an Activity (for example a Data Processing Request). After selecting a set of Production Requests the production planner will prompt the System Sheduling Interface (SSI) to schedule all the Data Processing Requests associated to the Data Processing. If possible, the activity will be scheduled on the appropriate resource and the System Resources Model will be notified of any changes to resources.

4.6.5.4.4.2 Interfaces With Other Subsystems and Segments

4.6.5.4.4.3 Stimulus

A production planner initiates the schedule function from the Production Planning Workbench.

4.6.5.4.4.4 Participating Classes From the Object Model

The following are participating classes from the Object Model:

- PlDSsi
- PlDSsiScheduler
- RSchRqst
- PlDSsiOppGen
- PlDSaAllocator
- SsiNoid

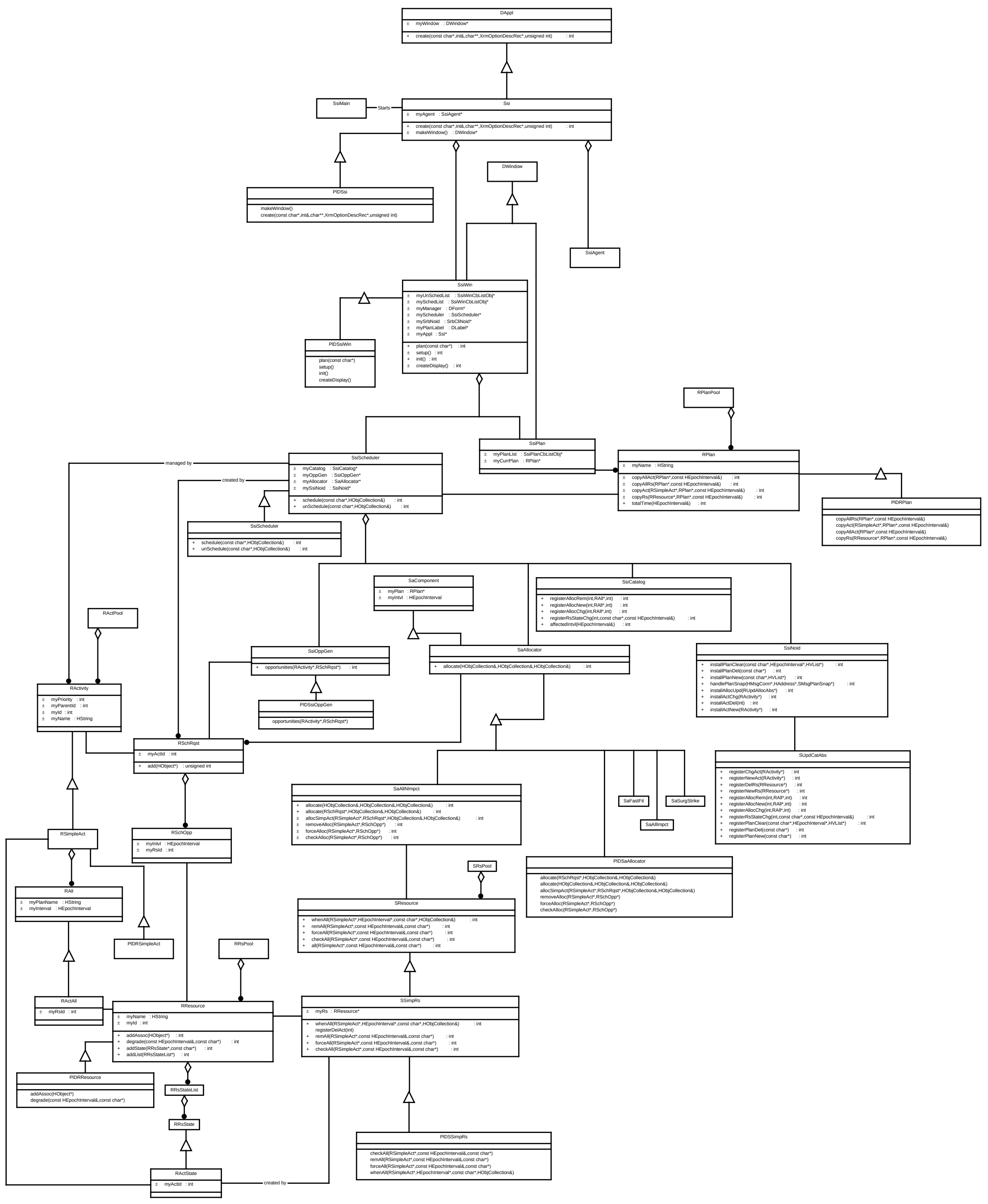
4.6.5.4.4.5 Beginning System, Segment and Subsystem State(s)

Steady state, up and running.

4.6.5.4.4.6 Ending State

No change in the state of the system.

Figure 4.6-6. Planning Workbench System Scheduling Interface


DAppl
± myWindow : DWindow*
+ create(const char*,int&,char**,XrmOptionDescRec*,unsigned int) : int
SsiMain
Starts
Ssi
± myAgent : SsiAgent*
+ create(const char*,int&,char**,XrmOptionDescRec*,unsigned int) : int
± makeWindow() : DWindow*
PlDSsi
makeWindow()
create(const char*,int&,char**,XrmOptionDescRec*,unsigned int)
DWindow
SsiAgent
SsiWin
± myUnSchedList : SsiWinCbListObj*
± mySchedList : SsiWinCbListObj*
± myManager : DForm*
± myScheduler : SsiScheduler*
± mySrbNoid : SrbCliNoid*
± myPlanLabel : DLabel*
± myAppl : Ssi*
+ plan(const char*) : int
± setup() : int
+ init() : int
± createDisplay() : int
PlDSsiWin
plan(const char*)
setup()
init()
createDisplay()
RPlanPool
SsiPlan
± myPlanList : SsiPlanCbListObj*
± myCurrPlan : RPlan*
RPlan
± myName : HString
± copyAllAct(RPlan*,const HEpochInterval&) : int
± copyAllRs(RPlan*,const HEpochInterval&) : int
± copyAct(RSimpleAct*,RPlan*,const HEpochInterval&) : int
± copyRs(RResource*,RPlan*,const HEpochInterval&) : int
+ totalTime(HEpochInterval&) : int
PlDRPlan
copyAllRs(RPlan*,const HEpochInterval&)
copyAct(RSimpleAct*,RPlan*,const HEpochInterval&)
copyAllAct(RPlan*,const HEpochInterval&)
copyRs(RResource*,RPlan*,const HEpochInterval&)
managed by
created by
SsiScheduler
± myCatalog : SsiCatalog*
± myOppGen : SsiOppGen*
± myAllocator : SaAllocator*
± mySsiNoid : SsiNoid*
+ schedule(const char*,HObjCollection&) : int
+ unSchedule(const char*,HObjCollection&) : int
SsiScheduler
+ schedule(const char*,HObjCollection&) : int
+ unSchedule(const char*,HObjCollection&) : int
SaComponent
± myPlan : RPlan*
± myIntvl : HEpochInterval
SsiCatalog
+ registerAllocRem(int,RAll*,int) : int
+ registerAllocNew(int,RAll*,int) : int
+ registerAllocChg(int,RAll*,int) : int
+ registerRsStateChg(int,const char*,const HEpochInterval&) : int
+ affectedIntvl(HEpochInterval&) : int
RActPool
SsiOppGen
+ opportunities(RActivity*,RSchRqst*) : int
PlDSsiOppGen
opportunities(RActivity*,RSchRqst*)
SaAllocator
+ allocate(HObjCollection&,HObjCollection&,HObjCollection&) : int
SsiNoid
+ installPlanClear(const char*,HEpochInterval*,HVList*) : int
+ installPlanDel(const char*) : int
+ installPlanNew(const char*,HVList*) : int
+ handlePlanSnap(HMsgConn*,HAddress*,SMsgPlanSnap*) : int
+ installAllocUpd(RUpdAllocAbs*) : int
+ installActChg(RActivity*) : int
+ installActDel(int) : int
+ installActNew(RActivity*) : int
RActivity
± myPriority : int
± myParentId : int
± myId : int
± myName : HString
RSchRqst
± myActId : int
+ add(HObject*) : unsigned int
SUpdCatAbs
+ registerChgAct(RActivity*) : int
+ registerNewAct(RActivity*) : int
+ registerDelRs(RResource*) : int
+ registerNewRs(RResource*) : int
+ registerAllocRem(int,RAll*,int) : int
+ registerAllocNew(int,RAll*,int) : int
+ registerAllocChg(int,RAll*,int) : int
+ registerRsStateChg(int,const char*,const HEpochInterval&) : int
+ registerPlanClear(const char*,HEpochInterval*,HVList*) : int
+ registerPlanDel(const char*) : int
+ registerPlanNew(const char*) : int
SaAllNImpct
+ allocate(HObjCollection&,HObjCollection&,HObjCollection&) : int
+ allocate(RSchRqst*,HObjCollection&,HObjCollection&) : int
± allocSimpAct(RSimpleAct*,RSchRqst*,HObjCollection&,HObjCollection&) : int
± removeAlloc(RSimpleAct*,RSchOpp*) : int
± forceAlloc(RSimpleAct*,RSchOpp*) : int
± checkAlloc(RSimpleAct*,RSchOpp*) : int
SaFastFit
SaSurgStrike
SaAllImpct
RSimpleAct
RSchOpp
± myIntvl : HEpochInterval
± myRsId : int
PlDSaAllocator
allocate(RSchRqst*,HObjCollection&,HObjCollection&)
allocate(HObjCollection&,HObjCollection&,HObjCollection&)
allocSimpAct(RSimpleAct*,RSchRqst*,HObjCollection&,HObjCollection&)
removeAlloc(RSimpleAct*,RSchOpp*)
forceAlloc(RSimpleAct*,RSchOpp*)
checkAlloc(RSimpleAct*,RSchOpp*)
SRsPool
RAll
± myPlanName : HString
± myInterval : HEpochInterval
SResource
+ whenAll(RSimpleAct*,HEpochInterval*,const char*,HObjCollection&) : int
+ remAll(RSimpleAct*,const HEpochInterval&,const char*) : int
+ forceAll(RSimpleAct*,const HEpochInterval&,const char*) : int
+ checkAll(RSimpleAct*,const HEpochInterval&,const char*) : int
+ all(RSimpleAct*,const HEpochInterval&,const char*) : int
PlDRSimpleAct
RRsPool
RActAll
± myRsId : int
RResource
± myName : HString
± myId : int
+ addAssoc(HObject*) : int
+ degrade(const HEpochInterval&,const char*) : int
+ addState(RRsState*,const char*) : int
+ addList(RRsStateList*) : int
SSimpRs
± myRs : RResource*
+ whenAll(RSimpleAct*,HEpochInterval*,const char*,HObjCollection&) : int
registerDelAct(int)
+ remAll(RSimpleAct*,const HEpochInterval&,const char*) : int
+ forceAll(RSimpleAct*,const HEpochInterval&,const char*) : int
+ checkAll(RSimpleAct*,const HEpochInterval&,const char*) : int
PlDRResource
addAssoc(HObject*)
degrade(const HEpochInterval&,const char*)
RRsStateList
RRsState
PlDSSimpRs
checkAll(RSimpleAct*,const HEpochInterval&,const char*)
remAll(RSimpleAct*,const HEpochInterval&,const char*)
forceAll(RSimpleAct*,const HEpochInterval&,const char*)
whenAll(RSimpleAct*,HEpochInterval*,const char*,HObjCollection&)
RActState
± myActId : int
created by

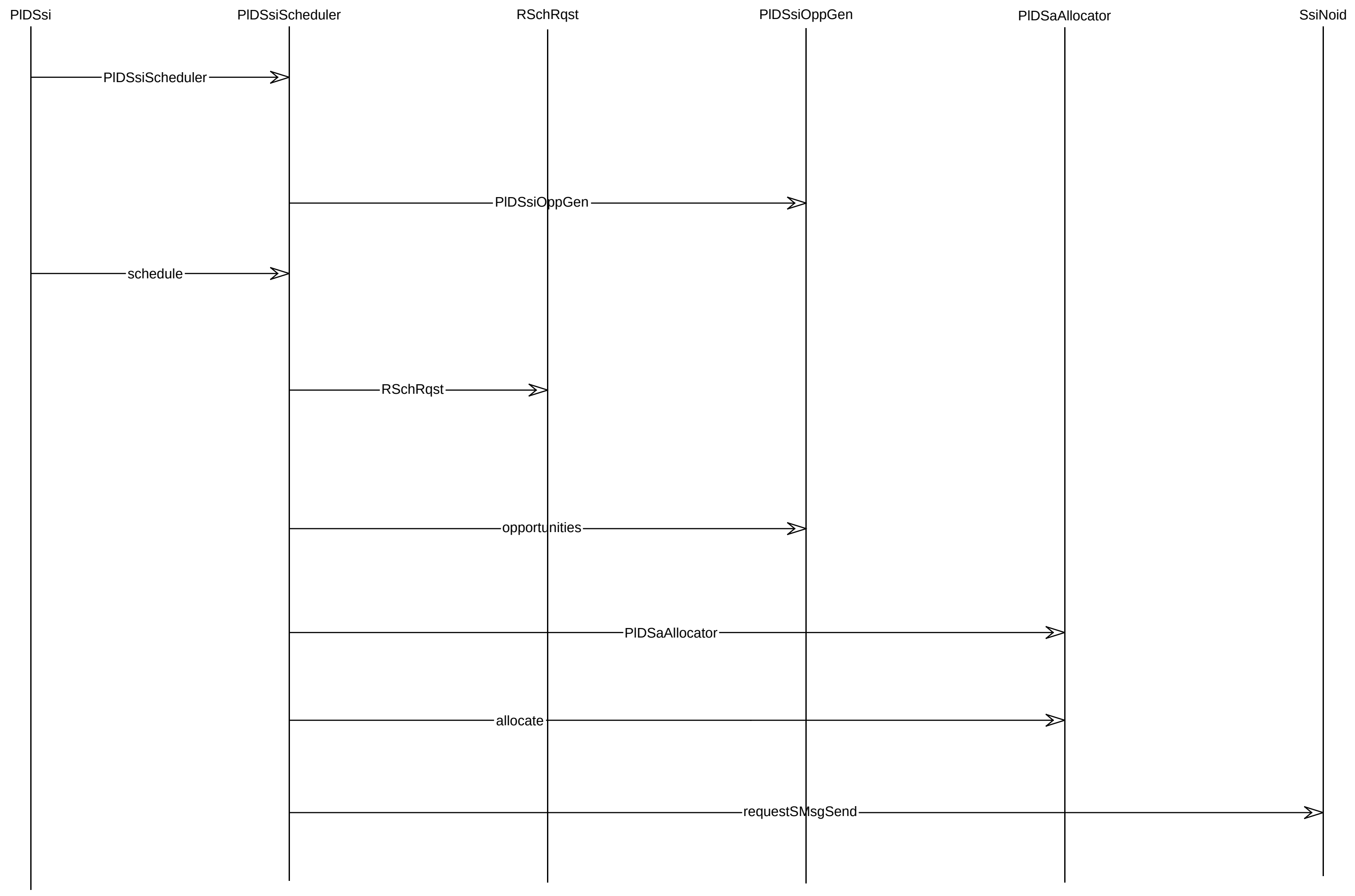
4.6.5.4.4.7 Scenario Description

1. The production planner starts the planning workbench utility, the standard user authentication process applies (see scenario 4.5.17).
2. An PlDSsiScheduler is created
3. The PlDSsiScheduler iterates through an ordered list of activities to be scheduled.
3. The production planner selects an activity to be scheduled which is sent to PlDSsiScheduler. .
4. The PlDSsiScheduler creates an instance of the Scheduling Request (RSchRqst) which corresponds to the selected activity (RSimpleAct).
5. The PlDSsiScheduler creates an instance of an Opportunity Generator (PlDSsiOppGen).
6. The PlDSsiScheduler passes the Scheduling Request (RSchRqst) and the selected activity to the Opportunity Generator.
7. The Opportunity Generator finds a resource on which the selected activity can be scheduled. This resources information is stored in the Scheduling Requests (RSchRqst).
8. The PlDSsiScheduler creates an instance of a Resource Allocator (PlDSaAlloctor).
9. The PlDSsiScheduler passes the Scheduling Request to the Resource Allocator and asks the Resource Allocator to allocate the activity from the resource information stored in the Scheduling Request.
10. The PlDSsiScheduler notifies the System Resource Model (through the SsiNoid) of the changes to the resource on which the activity is scheduled.

4.6.5.4.4.8 Event Trace

See Figure 4.6-7.

Figure 4.6-7. Scheduling an Activity Event Trace


PlDSsi
PlDSsiScheduler
RSchRqst
PlDSsiOppGen
PlDSaAllocator
SsiNoid
PlDSsiScheduler
PlDSsiOppGen
schedule
RSchRqst
opportunities
PlDSaAllocator
allocate
requestSMsgSend